

Т.Э. Кренкель
А.Г. Коган
А.М. Тараторин

Персональные ЭВМ

в инженерной
практике

· Радио и связь ·

Т.Э. Кренкель
А.Г. Коган
А.М. Тараторин

Персональные ЭВМ в инженерной практике



Москва
„Радио и связь“
1989

ББК 32.972
К79
УДК 681.322—181 4.004.14

Рецензент: д-р техн. наук, профессор И. М. Витенберг

Редакция литературы по вычислительной технике
Кренкель Т. Э. и др.

К79 Персональные ЭВМ в инженерной практике: Справочник/Т. Э. Кренкель, А. Г. Коган, А. М. Тараторин. — М.: Радио и связь, 1989. — 336 с.: ил.

ISBN 5-256-00531-6.

Книга является специализированным справочником, посвященным прикладным аспектам использования персональных ЭВМ (ПЭВМ). В нем отражена специфика работы с конкретными ПЭВМ, совместимыми с IBM PC: ЕС 1840, ЕС 1841, Искра 1030, Нейрон и др. Изложены основные правила работы с ПЭВМ, принципы создания и использования современного программного обеспечения. Приведен обзор пакетов прикладных программ для решения различных научно-технических задач. Рассмотрены особенности программирования ПЭВМ на языках Паскаль, Си, Бейсик, Фортран.

Для пользователей ПЭВМ и программистов.

К 2404000000-125 КБ 59-12-88
046(01)—89

ББК 32.972

Справочное издание

Кренкель Теодор Эрнестович

Коган Андрей Горацевич

Тараторин Александр Маркович

ПЕРСОНАЛЬНЫЕ ЭВМ В ИНЖЕНЕРНОЙ ПРАКТИКЕ

Справочник

Заведующая редакцией Г. И. Козырева

Редактор Т. М. Бердичевская

Обложка художника В. Ф. Громова

Художественный редактор Н. С. Шейн

Технический редактор Т. Н. Зенкина

Корректор Н. Л. Жукова

ИБ № 1898

Сдано в набор 25.10.88		Подписано в печать 25.04.89	
Т-09946	Формат 60×90 1/16	Бумага тип № 3	Гарнитура литературная
	Печать офсетная	Усл. печ. л. 21,0	Усл. кр.-отт. 21,37
Уч.-изд. л. 22,15	Тираж 170 000 экз	(1 завод 1-70000 экз.) Издательство «Радио и связь»	
	Изд. № 22512	Зак. № 1439	Цена 1 р. 10 к.

Издательство «Радио и связь» 101000 Москва, Почтамт, а/я 693
Типография Прейскурантиздата 125438 Москва, Пакгаузное шоссе, 1

ISBN 5-256-00531-6

© Издательство «Радио и связь», 1989

ПРЕДИСЛОВИЕ

Сейчас на наших глазах совершается компьютерная революция. Почему? Во-первых, вместо больших ЭВМ, которые можно было бы сравнить с маленькими островками автоматизации в безбрежном море проблем, появляются компактные, чуть больше пишущей машинки персональные ЭВМ (ПЭВМ) или персональные компьютеры (ПК). Во-вторых, эти компьютеры по возможностям уже не уступают хорошей машине десятилетней давности, требующей специальных залов и множества операторов в белых халатах. В-третьих, как мы надеемся, скоро цена ПЭВМ станет доступной любому инженеру. В-четвертых, подавляющее большинство современных ПЭВМ совместимо между собой: программа, написанная любым из десятков миллионов ее пользователей во всем мире, без переделок заработает и на Вашей ПЭВМ. В-пятых, все ресурсы компьютера, в том числе и высококачественная цветная графика — в полном распоряжении счастливого владельца. В-шестых, многомиллионное распространение ПЭВМ произвело настоящую революцию в программном обеспечении: оно становится все более простым, удобным, изящным, не говоря уже о том, что теперь на рынке программных средств можно найти почти все что угодно — от программ для поддержания оптимального режима выращивания океанических рыб в искусственных условиях до сложнейших систем автоматизированного проектирования с элементами искусственного интеллекта. В-седьмых...

Достоинства ПЭВМ можно перечислять еще долго. Несомненно, что они впервые реально позволяют повсеместно автоматизировать человеческую деятельность и высвободить интеллектуальные ресурсы для творческого труда.

В настоящее время большое внимание уделяется внедрению ПЭВМ в производство, инженерные и научные исследования. Для решения этой задачи используется широкий спектр ПЭВМ, совместимых с IBM PC: ЕС 1840, ЕС 1841, Искра 1030, Нейрон и др. Совместимые компьютеры выпускаются также и в странах СЭВ (ПНР, НРБ, ВНР).

В связи с широким внедрением ПЭВМ возникает необходимость в своеобразной «минимизации» затрат времени на освое-

ние этой техники максимально широким кругом пользователей. Эта проблема, на наш взгляд, является одной из основных и вот почему: индустрия современных ПК превратилась в целую отрасль знания. Если подсчитать количество технических описаний, необходимых только для грамотного использования аппаратных средств ПЭВМ, понадобится не менее полутора десятков увесистых книг. С прикладным программным обеспечением ситуация вообще катастрофическая: «банк» прикладных пакетов программ для ПК составляет не менее 50 тысяч наименований. Хотелось бы отметить, что подавляющее большинство этих программ сложны, объемны и для их изучения требуются специальные руководства. В связи с этим пользователь ПЭВМ часто оказывается в положении человека, не знающего, с чего ему начать — возникает своеобразный психологический шок, вызванный огромным разнообразием предлагаемых средств.

В нашей стране и за рубежом издано много книг, посвященных различным аспектам использования ПК, например [1—3]. Однако большинство этих публикаций обращены либо к широкому кругу начинающих пользователей ПЭВМ, которые будут использовать компьютер преимущественно для подготовки текстов или для обработки деловой информации, либо к программистам «только на Бейсике», либо к специалистам по архитектуре ПЭВМ. Из этого круга лиц «выпали» специалисты инженерного профиля. Таким специалистам, на наш взгляд, нужно в краткой форме изложить довольно разноплановую информацию: начать с простейших методов работы с ПЭВМ и рассказать о существующих пакетах прикладных программ, а затем более детально описать функции работы с операционной системой, возможности управления устройствами ПЭВМ и особенности программирования ПК на различных языках высокого уровня. В идеале должен бы был получиться своеобразный «путеводитель» по использованию современной ПЭВМ, который бы в равной степени пригодился как начинающему пользователю, так и специалисту более высокой квалификации.

В этой книге мы предприняли именно такую попытку. На наш взгляд, она может пригодиться как начинающим, так и квалифицированным пользователям и программистам. Главы книги относительно независимы.

Глава 1 содержит минимальный набор сведений, необходимых для того, чтобы уметь включать компьютер, выполнять несколько простейших функций операционной системы ДОС и запускать программу. Прочтя эту главу, читатель без особых трудностей сможет запустить любую из прикладных программ и работать с ней.

В главе 2 представлен обзор современного прикладного программного обеспечения. Здесь предпринята попытка описать наиболее характерные черты различных пакетов программ,

а не конкретные пакеты — это просто невозможно. На наш взгляд, прочтя любой параграф этой главы, читатель должен понять принципиальные возможности соответствующего программного обеспечения и наиболее типичные приемы работы с ним на ПЭВМ. Это должно помочь ему ориентироваться в современном программном обеспечении.

В гл. 3 работа с операционной системой ДОС рассматривается более детально. Эта глава понадобится любому пользователю при систематической работе на компьютере.

Главы 4—6 посвящены принципам программирования ПК. Заметим, что их надо читать в такой последовательности: сначала гл. 4 и 5, в которых с общих позиций рассмотрены принципы построения программ и управления устройствами ПЭВМ на чрезвычайно простом абстрактном «языке» программирования, затем гл. 6, где дается «перевод» конструкций этого языка на языки Паскаль, Си и Бейсик. Сразу же заметим: шестая глава не сможет заменить руководств по указанным языкам программирования: это одновременно элементарное введение в язык (отчасти проникнутое авторским отношением), знакомство с тем, как устроены компиляторы языков на ПЭВМ, и краткий обзор функций управления устройствами ПЭВМ с данного языка. При этом изложение ведется единым образом: вначале дан перевод основных программных конструкций, а затем описаны функции управления внешними устройствами ПЭВМ. На наш взгляд, такой подход значительно облегчает изучение языков программирования. Хотелось бы обратить внимание на то, что мы постарались описать возможно более полный набор команд для управления периферийными устройствами ПЭВМ на различных языках программирования. Эти команды пригодятся в равной степени всем — и начинающим и профессионалам.

Мы хотели бы, чтобы эта книга стала помощником для инженеров, применяющих ПЭВМ в своей работе. Конечно, она не сможет заменить многочисленные руководства по прикладным пакетам программ, операционной системе, языкам программирования. Она должна лишь показать реальные возможности ПЭВМ при решении различных задач и создать мотивацию для более углубленного изучения конкретных разделов.

Глава 1. ВВЕДЕНИЕ В ПЕРСОНАЛЬНУЮ ЭВМ

Упоминание о компьютеризации как о черте современной жизни стало уже привычным. Увеличив на несколько порядков быстродействие и память, уменьшившись за пару десятилетий от размеров динозавра до размеров вируса, компьютер проникает во все сферы человеческой деятельности. То и дело приходится слышать, что к решению той или иной задачи пора подойти «на современном научном уровне с применением ЭВМ». Поэтому естествен интерес непосвященного к поистине волшебному устройству, одно применение которого якобы обеспечивает научность подхода к интересующей его проблеме.

Первое же знакомство с компьютером открывает совершенно особый мир, мир безграничных возможностей, полностью послушный человеку, держащему руку на клавиатуре. Выясняется, впрочем, что в отличие от древнего джина, готового слушаться и повиноваться сразу же по выходе из бутылки, современный электронный джин предлагает нам сначала в эту бутылку влезть — т. е. научиться говорить (а может быть и думать?) на его языке, изучить все его особенности и прихоти, даже овладеть средствами борьбы с ним в некоторых специальных случаях. И хотя каждый том многотомной документации убеждает нас, что все устроено просто и бесхитростно, давно известный закон срabатывает безотказно и именно наш с Вами случай как раз и оказывается специальным, требующим досконального изучения внутреннего мира компьютера. Не случайно, должно быть, одна из самых известных книг про ПК так и называется: «ПК. Взгляд изнутри» [28].

Остановимся на мгновение и спросим себя: а стоит ли растрачивать собственные, в общем-то довольно скромные силы на изучение изнутри нашего замысловатого электронного помощника, не будучи уверенным, что тот действительно в состоянии нам помочь? Чего нового могут ожидать от компьютера инженер, экспериментатор, проектировщик?

Авторы относят себя к разряду пользователей, т. е. специалистов в своей конкретной области, которых компьютер интересует как инструмент решения своих задач. Так, мы привыкли рассматривать молоток как средство для забивания гвоздей, а не как объект для изучения (изучению подлежит разве что техника работы молотком). Точно так же мы склонны относиться и к ПК и даже к ускорителю на встречах пучках. И ПК манит нас вслед за собой: глаза разбегаются от изумительно красивых прикладных пакетов программ, научиться работать с которыми можно не влезая в дебри устройства компьютера.

Быть может, Вам повезет и будет обнаружен идеально подходящий пакет прикладных программ, работа с которым удовлетворит Вас на несколько лет вперед. Более вероятно,

однако, что компьютер лишь поманит Вас за собой (так случилось и с нами): вы обнаружите прекрасное математическое обеспечение, которое очень близко к тому, что Вам жизненно необходимо, но все же Вам не подходит. Тогда, замороженный близкой и осязаемой возможностью создать именно то, что нужно, Вы незаметно погрузитесь «внутри» ПК. Мы надеемся, что наша книга поможет Вам избежать некоторых типичных подводных камней и ускорит движение к конечной цели.

1.1. КАК ПЭВМ ВЫГЛЯДИТ И ИЗ ЧЕГО СОСТОИТ

ПЭВМ представляет собой вычислительную систему, состоящую из системного блока, расположенного в одном или в двух корпусах, дисплея и клавиатуры. К этому базовому комплекту могут быть подключены разнообразные дополнительные устройства, например принтер, на котором можно распечатывать рисунки и тексты программ. Внешний вид типичной ПЭВМ показан на рис. 1.1. В том виде, в котором она показана на фотографии, ПЭВМ является результатом огромного прогресса в вычислительной технике за последнее десятилетие.



Рис. 1.1. Персональная ЭВМ

За последние пять—семь лет в вычислительной технике произошла стремительная переоценка ценностей, через которую прошли и авторы. Для того чтобы читатель понял, о чем идет речь, приведем один пример. В 1982 г. один из авторов получил возможность работать на компьютере, который в то время считался верхом совершенства. Это была 16-разрядная мини-ЭВМ с оперативной памятью 128 Кбайт на пользователя, имеющая два внешних диска по 5 Мбайт каждый. К этой

системе было подключено 4 терминала, один из которых позволял строить графики. Системный блок компьютера по размерам соответствовал большому холодильнику.

Компьютер был весьма чувствителен к нестабильности сети, запыленности и требовал для своей работы отдельного кондиционированного помещения. Эта система вызвала восторженную реакцию у научных сотрудников, которые впервые получили возможность работать на компьютере непосредственно в исследовательской лаборатории. Стоимость системы составляла около 100 тыс. руб. В 1983 г. одного из авторов попросили дать консультацию по закупке компьютера (позднее выяснилось, что это была одна из первых моделей 16-разрядной ПЭВМ). Параметры этой системы, судя по краткому описанию, не уступали только что описанным, но стоимость была крайне низкой. Такое соотношение параметров ЭВМ и ее цены было трудно допустить, поэтому автор отнесся к рекламному проспекту более чем скептически и предположил, что либо в каталоге в графе стоимости где-то потерян ноль, либо параметры системы не соответствуют указанным в проспекте. Но уже через год стало понятно, что ПК может стоить в сто раз меньше типичных мини-ЭВМ и при этом иметь намного превосходящие их возможности.

В чем основные преимущества использования ПК? Во-первых, современная ПЭВМ, совместимая с распространенными моделями класса IBM PC XT (советские ЭВМ ЕС 1840, ЕС 1841, Искра 1030, Нейрон), по параметрам не уступает средней ЭВМ пяти—десятилетней давности. Это означает, что в распоряжение пользователя попадает компьютер с весьма серьезными возможностями. В то же время ПЭВМ компактна, легко переносится, не требует кондиционирования и специального помещения. Во-вторых, эта ЭВМ является персональной в полном смысле этого слова: она имеет относительно низкую стоимость и все ее ресурсы (память, диски, графика, печатающие устройства) всегда в полном распоряжении ее владельца. Это имеет очень важное значение: все вычислительные возможности компьютера всегда под рукой и их не надо делить между пользователями. В-третьих, для ПЭВМ разработано огромное математическое обеспечение, не идущее ни в какое сравнение с математическим обеспечением средних и больших ЭВМ как по разнообразию, так и по удобству работы. В известном смысле именно появление ПЭВМ произвело революцию в программном обеспечении современных компьютеров. Рассмотрим основные компоненты ПК.

Системный блок содержит всю электронную часть ПЭВМ, блок питания и устройства для хранения информации на внешних магнитных носителях. По размерам он напоминает обычный проигрыватель. Внутри блока содержится сердце компьютера — процессор и оперативная память. В описываемой ПЭВМ исполь-

зуется 16-разрядный микропроцессор типа K1810BM86. Тактовая частота процессора стандартной конфигурации примерно 5 МГц. Типичная команда ЭВМ (сложение целых чисел) выполняется за 25—30 тактов процессора, а умножение за 130—150 тактов. Это означает, что средняя производительность ПЭВМ составляет около 100 тыс. операций в секунду. Максимальная емкость оперативной памяти может достигать 640 Кбайт. Минимальная емкость оперативной памяти 64 Кбайт, но такая память, безусловно, мала для серьезной работы с большими пакетами программ.

Внутреннее устройство системного блока может быть различным. Наиболее распространен следующий стандарт: собственно процессор и оперативная память собраны на так называемой системной плате компьютера, которая расположена у дна системного блока. На этой плате имеется несколько разъемов, в которые могут быть вставлены стандартные платы расширения: устройства управления дисплеем, интерфейсы дисков, дополнительная память и др. Встречается и другая конструкция системного блока: системная плата отсутствует, а весь компьютер собран на основе набора стандартных плат, вставляемых в специальные разъемы системного блока.

В системном блоке или в дополнительном блоке расположены *накопители* на гибких и фиксированных магнитных дисках (НГМД и НМД). Гибкие диски (распространены также названия флоппи-диски или дискеты) имеют диаметр 5,25 дюйма (133 мм) и заключены в специальный конверт с прорезью (рис. 1.2). Для их считывания используется специальное устройство — дисковод, у которого имеется специальная шторка. Для того чтобы вставить дискету в компьютер, необходимо вложить ее в прорезь дисковода и закрыть эту шторку. Стандартная емкость гибкого диска 360 Кбайт. Это означает, что на нем можно записать 360 тысяч знаков или примерно 200 стандартных страниц текста. На гибких дисках записываются и распространяются все прикладные программы для ПЭВМ: операционная система, редакторы текста, игры, пакеты программ и т. д. В одном компьютере может использоваться один или два дисковода для гибких дисков. Первый из них всегда обозначается буквой А, а второй — буквой В.

Основной фиксированного диска служит жесткий металлический диск, на который информация записывается по специальной технологии, позволяющей уплотнить запись примерно в 100 раз. Первоначально такие диски были выполнены как два диска по 30 Мбайт каждый и обозначались цифрами 30/30 подобно калибру охотничьего ружья «Винчестер». Это название «приклеилось» ко всем фиксированным дискам и теперь принято говорить «Винчестерский диск» или просто «Винчестер». Внешне винчестер представляет собой маленькую черную плоскую коробочку. На ПЭВМ используются самые разнообразные типы



Рис. 1.2. Гибкий диск

фиксированных дисков. Обычно емкость такого диска превышает 5 Мбайт. Стандартной для ПЭВМ является емкость диска 10 Мбайт. Встречаются фиксированные диски и большой емкости: 40, 60, 120 и даже 300 Мбайт. Фиксированные диски имеют небольшие размеры и герметично изолированы от внешней среды, следовательно, не требуют специальных обеспыленных помещений. Однако фиксированные диски являются довольно деликатным устройством и могут испортиться из-за резких механических воздействий на ПЭВМ, поэтому работающий компьютер не рекомендуется передвигать или переносить. На ПЭВМ может быть установлено несколько фиксированных дисков. Они всегда обозначаются буквами, начиная с буквы С латинского алфавита (D, E и т. д.).

Дисплей ПК является основным устройством для отображения информации, выводимой во время работы программ. Внешне дисплей выглядит почти как обычный малогабаритный телевизор. В составе ПЭВМ используется несколько типов дисплеев. В минимальной конфигурации ПЭВМ применяется алфавитно-цифровой дисплей, позволяющий выводить на экран только буквы и цифры в одном цвете. Основное преимущество работы с ПК, однако, в использовании графического представления информации. Графические дисплеи различаются по своему разрешению и бывают черно-белыми или цветными. Для каждого типа дисплея используется свой адаптер, т. е. специальное устройство управления отображением информации, находящееся в системном блоке. Стандартным для ПЭВМ является цветной дисплей, имеющий следующие характеристики:

в текстовом режиме разрешение 80 символов в строке и 25 символов в столбце (80×25) или разрешение 40×25 ;

в черно-белом графическом режиме разрешение 640×200 точек (два уровня яркости);

в цветном режиме разрешение 320×200 точек и одновременное отображение четырех цветов из палитры 16 цветов.

В последнее время получают распространение альтернативные графические системы, имеющие лучшие характеристики. Например, адаптер улучшенной графики позволяет визуализировать 640×350 точек в 16 цветах из набора 64 цвета. Выбор дисплея и графического адаптера очень важен, так как при работе с низким графическим разрешением у пользователя компьютера быстро устают глаза. Так, если основным применением компьютера является обработка текстовой информации, то желательно приобрести не графический, а монохромный алфавитно-цифровой дисплей.

Клавиатура ПЭВМ представляет собой стандартное устройство, предназначенное для ввода команд и данных (рис. 1.3). Типичные размеры клавиатуры $40 \times 450 \times 180$ мм. На клавиатуре имеется несколько групп клавиш, с которыми необходимо познакомиться каждому пользователю ПЭВМ, так как они используются в различных программах. Отметим, что в различных версиях клавиатуры эти клавиши могут быть расположены не совсем так, как это изображено на рисунке. В центре клавиатуры расположены алфавитно-цифровые клавиши для ввода букв и цифр. Клавиша, расположенная в правой части алфавитной клавиатуры, служит для возврата каретки. В дальнейшем будем называть ее «Возврат каретки» или «Ввод». Нажатием на нее переводится строка текста или заканчивается ввод числовых данных. Над ней расположена клавиша, на которой обозначена стрелка. Эта клавиша используется для стирания ближайшего слева символа и называется также «Забой». При нажатии на нее текущая позиция смещается влево. Под клавишей «Возврат каретки» в правой и левой части клавиатуры расположены клавиши «Верхний регистр», переключающие

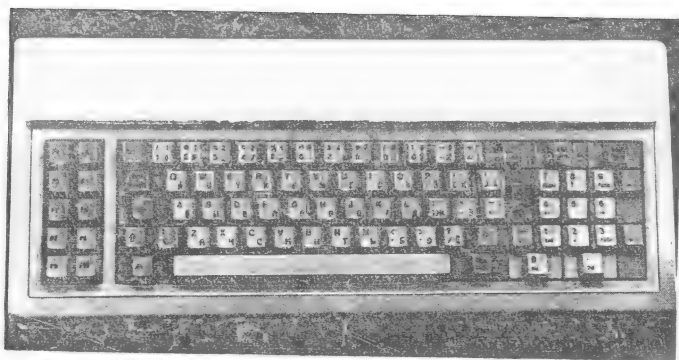


Рис. 1.3. Клавиатура ПЭВМ

режимы маленьких и больших букв, как и на обычной пишущей машинке. Режим работы в верхнем регистре можно зафиксировать, нажав на клавишу CapsLock и выключить еще одним нажатием на эту клавишу. Длинная клавиша в нижней части клавиатуры служит для ввода пробела. В левой части алфавитной группы клавиш расположены (сверху вниз):

клавиша Esc (спасение, выход, переход). Она используется в различных программах для прекращения работы программы;

клавиша табуляции, позволяющая переводить текущую позицию курсора на заданное число шагов вправо или влево;

клавиша Ctrl (управление) обычно используется одновременно с другими клавишами. При этом генерируется ряд специфических кодов, имеющих различный смысл в различных программах. Например, команда Ctrl-C (одновременно нажаты клавиши Ctrl и C) прерывает выполнение программы.

клавиша Alt (изменение, дополнение) аналогична клавише Ctrl и используется для генерации различных кодов. Например, в одном из редакторов текста одновременное нажатие клавиши Alt и цифры соответствует переходу в другое экранное окно.

В левой части клавиатуры расположены десять *функциональных клавиш*, обозначаемых F1—F10. Они используются в различных программах и системах для выполнения различных функций, например для выбора раздела меню. Простейшие примеры: при работе с операционной системой (ОС) нажатие на клавишу F3 повторяет последнюю набранную на клавиатуре команду, при работе с интерпретатором Бейсика клавиша F1 позволяет просмотреть на экране текст программы.

Группа клавиш в правой части клавиатуры имеет двойное назначение. Клавиши, обозначенные стрелками, используются для перемещения курсора по тексту (вверх, влево, вправо, вниз). Кроме того, имеются клавиши: PgUp (страница вверх) и PgDn (страница вниз) — для пролистывания страниц. Клавиши Home и End используются различными программами чаще всего для того, чтобы курсор вернулся на первую и последнюю позиции текста в строке. В то же время эти клавиши могут служить для ввода числовых данных, если нажать на специальную клавишу NumLock (включение числовой клавиатуры). Справа от этой клавиши расположена клавиша «Стоп» или Break, которая обычно используется вместе с клавишей Ctrl. Одновременное нажатие этих клавиш прерывает исполнение большинства программ. Снизу расположены две очень важные клавиши: Ins (вставка) и Del (стирание). Клавиша Ins обычно переключает режимы работы текстовых редакторов, а клавиша Del позволяет стереть символ, расположенный справа от текущей позиции текста. В то же время эти клавиши используются и в сочетании с другими. Например, одновременное нажатие клавиш Ctrl-Alt-Del инициализирует ПЭВМ и перегружает ОС.

Вообще, по поводу использования клавиш ПЭВМ невозможно дать сколько-нибудь исчерпывающих рекомендаций: в различных программах используются их самые различные сочетания, и функции, выполняемые той или иной клавишей, могут радикально меняться.

Нормальный комплект ПЭВМ обязательно должен включать печатающее устройство. В настоящее время выпускаются самые разнообразные принтеры, имеющие различную скорость и качество печати. Наиболее распространены недорогие принтеры матричного типа. В этих принтерах изображения букв и цифр на бумаге формируются с помощью матрицы иглоочек 9×9 элементов, которые ударяют по красящей ленте. Скорость печати у таких принтеров различна и колеблется от 12 до 200 символов в секунду.

К ПК можно подключить также разнообразные дополнительные устройства: накопители на магнитной ленте, графопостроители, специальные устройства для обмена информацией, системы ввода и вывода аналоговых сигналов и многие другие. Из наиболее распространенных устройств можно упомянуть систему «мышь» (mouse). Мышь представляет собой небольшую коробочку, которую можно передвигать по плоской поверхности. Относительные перемещения «мышки» считываются компьютером и могут быть преобразованы в движение курсора по экрану. Это очень удобно при выполнении на ЭВМ чертежных работ, управлении меню. Например, в ряде программ для того чтобы выполнить различные функции, необходимо, двигая «мышку», установить курсор на то место экрана, в котором отображен необходимый раздел меню, и нажать на кнопку, установленную на «мышке».

Все перечисленные устройства подключаются к системному блоку с помощью соединительных кабелей. Как начать работу на ПЭВМ? Прежде всего необходимо подключить к ней все устройства, руководствуясь техническим описанием. Затем необходимо взять гибкий диск с операционной системой ДОС, являющейся стандартом для ПЭВМ и вставить его в дисковод. Включим питание. После включения компьютер выполняет программы внутреннего тестирования. Как правило, при этом компьютер должен выдать звуковой сигнал, а на экране появится какая-нибудь надпись типа «Тестирование памяти» и побегут цифры, например

10 Кбайт исправны

20 Кбайт исправны и т. д.

Через некоторое время на дисководе мигнет светодиод и будет слышен характерный звук, означающий, что информация считывается с гибкого диска. Это означает, что в данный момент происходит загрузка ДОС. Затем на экране появится надпись типа «ДОС-16, версия...» и, возможно, «Введите текущую дату и время». Наберем на клавиатуре число, месяц, год, время

и перед нами появится сочетание типа A>. Если теперь нажать на клавишу «Возврат каретки», то текущая строка на экране будет смещаться вниз. Мы находимся в операционной системе. Сочетание A> соответствует тому, что мы работаем с гибким диском. Если в компьютере имеется фиксированный диск, то, набрав на клавиатуре C:, мы перейдем на него и приглашение системы изменится на C>. Снова набрав A:, мы опять перейдем на флоппи-диск. Заметим, что если на фиксированном диске имеется ДОС, то вставлять гибкий диск в компьютер необязательно: система загрузится с фиксированного диска. Программа инициализации, однако, устроена таким образом, что первоначально компьютер всегда пытается обратиться к гибкому диску и, если он не находит там ДОС, программа загрузки ДОС автоматически обращается к фиксированному диску.

Тем читателям, у которых ПК под рукой и которым хочется сразу же получить зримый результат работы в ПЭВМ, мы рекомендуем обратиться к последнему примеру § 1.4, в котором описана последовательность действий при запуске программы на Бейсике. Более обстоятельному читателю мы советуем сразу же начать чтение следующих параграфов и только потом попытаться написать и выполнить свою первую программу.

1.2. ЧТО ТАКОЕ ДОС И ЗАЧЕМ ОНА НУЖНА

Для того чтобы обеспечить взаимодействие аппаратуры и программ ПК и дать пользователю возможность управлять его функциями, нужна операционная система.

Операционная система — это сложная и большая программа, имеющая много возможностей. Пользователь, работающий с ПК, до определенного момента может не задумываться над разнообразными функциями и структурой ОС; все, что ему нужно, — это уметь выполнять несколько простейших команд, необходимых для каждодневной работы. Целью этого параграфа является дать читателю именно такое поверхностное представление об ОС. После того как читатель прочтет эту главу, мы настоятельно советуем ему «поиграть» за клавиатурой ПЭВМ, выполняя простейшие команды, а через некоторое время обратиться к гл. 3, в которой мы более подробно рассмотрим структуру ДОС и опишем работу с так называемыми «настройками» над ОС.

Итак, после включения ПЭВМ и выполнения ею программ внутреннего тестирования она автоматически загрузит операционную систему ДОС. В некоторых случаях система запросит у пользователя текущее время и дату. После окончания загрузки ОС на экране ПК появится одно из сообщений типа «Операционная система ДОС-16, версия...». Теперь перед пользователем появится обозначение типа A> или C> и пустое поле экрана.

В этом режиме система ждет введения команд, причем мигающий курсор находится в первой позиции после фигурной стрелки. Буквы А или С указывают на текущий диск, с которым работает система: А соответствует флоппи-диску, а С — фиксированному диску. Стрелка означает, что система ждет команды пользователя. Такое сочетание букв и стрелки называют приглашением ОС.

Зачем ДОС необходимо давать команды? Простейший вариант: системе нужно указать программу, которую она начнет выполнять. Для этого достаточно набрать на клавиатуре имя этой программы. Предположим, что программа называется LEXICON (название популярного текстового редактора). Наберем на клавиатуре LEXICPN, начиная писать буквы сразу после скобки А). Нажмем на клавишу «Возврат каретки». При этом мигнет лампочка на дисковом флоппи-диске и на экране появится сообщение «Плохая команда или имя файла» (в английском варианте «Bad command or file name»). И действительно, мы сделали грамматическую ошибку. Исправим ее. Для этого можно снова набрать на клавиатуре LEXICON. Другой путь — нажать на функциональную клавишу F3 (повтор команды). При этом на экране возникнет старое сочетание: А)LEXICPN. С помощью клавиши «Забой» сотрем два последних символа PN и допишем вместо них правильные. Снова дадим возврат каретки. Если редактор записан на флоппи-диске, то он загрузится в память, в противном случае повторится та же ситуация, что и при ошибке в написании названия команды.

Предположим, что редактор не загрузился. Что же делать пользователю в этой ситуации? По-видимому, редактор не записан на вставленном диске. Как это можно проверить?

Все команды, тексты, программы, графики записываются на диски в виде так называемых *файлов* — участков памяти на диске, имеющих свое имя. Таким файлом является и наш редактор. Для того чтобы попытаться его найти, самое простое действие — просмотреть список файлов на диске. Для этого в ОС имеется команда DIR (русский вариант КАТ) — просмотр каталога файлов. Наберем на клавиатуре DIR и дадим возврат каретки. Диск несколько раз мигнет и на экране появится список файлов, как это показано на рис. 1.4. Мы видим, что для каждого файла, хранящегося на диске, кроме его имени выдается информация о емкости в килобайтах, времени и дате создания.

Как видно из рисунка, на диске имеются самые разнообразные файлы: у них разные имена, а кроме того после имени некоторых файлов стоят сочетания из трех букв, называемые расширениями. При ссылке на имена файлов перед расширениями ставят точку. Зачем нужны эти странные расширения типа COM, TXT, EXE? Оказывается, каждое из них соответству-

SKC	COM	28160	10-29-86	10:34a
EXE2BIN	EXE	2816	3-07-85	1:43p
LINK	EXE	38144	3-07-85	1:43p
SKCOMM	COM	36318	9-16-85	4:01p
SKINST	COM	54656	1-01-83	
SK	HLP	53632	1-01-83	
SKINST	MSG	4224	1-01-83	
READ-ME	SK	12928	8-14-85	11:10a
BASIC	<DIR>		1-01-80	12:02a
VDISK	LST	136313	3-07-85	1:43p
TIMER	COM	1393	12-23-83	5:03p
PARK	COM	3947	4-10-86	9:20p
LISP	<DIR>		6-23-87	10:46a
DBASE	<DIR>		6-23-87	10:57a
AUTOEXEC	BAT	138	7-07-49	10:18a
CONFIG	SYS	45	6-24-87	9:35a
AUTOEXEC	BAK	130	6-25-87	11:18a
SHOW	<DIR>		6-25-87	11:26a
ACAD	<DIR>		6-25-87	11:31a
LOTUS	<DIR>		6-30-87	7:05p
UTIL	<DIR>		7-12-87	12:26p
FIG	<DIR>		7-15-87	1:57p
		66 File(s)	647168 bytes free	

C:\>

Рис. 1.4. Экран ПЭВМ в режиме вывода списка файлов

ет типу информации, записанному в файл. Например, расширение .TXT соответствует тому, что в этом файле записан текст. Встречаются также расширения типа .DOC (от слова документ). Расширение .EXE соответствует тому, что в данном файле содержится код команды или программы, готовый к исполнению (от английского слова execute). Расширение .COM по смыслу аналогично .EXE. Кроме того, бывают расширения .OBJ (object) — объектный файл, получающийся в результате трансляции исходного текста программы, BAT — специальный тип файлов, содержащих набор команд в текстовом виде. Тексты программ, написанных на языках программирования, обычно имеют расширения .C (язык Си), .BAS (Бейсик), .FOR (Фортран), .PAS (Паскаль). Необходимо обратить внимание читателя на то, что некоторые расширения могут быть произвольными. Например, создавая текстовый файл, читатель может присвоить ему расширение из первых трех букв своей фамилии или вообще не задавать никакого расширения. ДОС, однако, устроена таким образом, что расширения .EXE, .COM и .BAT ей необходимы.

Но вернемся к поиску нашего редактора. Как видим, в имеющемся списке файлов его имени нет. Может быть, он записан на фиксированном диске? Напомним, что мы сейчас видим приглашение операционной системы A), что соответствует тому, что текущим каталогом является гибкий диск. Вообще, любой диск или каталог, в котором пользователь находится в данный момент, называется текущим: по умолчанию ОС проводит все операции именно в «текущей» среде, не обращая внимания на окружающее, если ей не дать специальные команды.

Для того чтобы сменить текущий диск, нам достаточно написать C: и дать возврат каретки. Приглашение изменилось и теперь мы видим на экране C>. Теперь можно снова дать команду на просмотр каталога. Напишем DIR и увидим, что список имен файлов на экране пробегает с довольно большой скоростью и скрывается в верхней части экрана. Это связано с тем, что на фиксированном диске файлов много и все они не помещаются одновременно на экране, поэтому система «пролистывает» список перед нами.

Для того чтобы остановить пробегающий список файлов, в системе предусмотрена специальная функция, выполнить которую можно нажав одновременно клавиши Ctrl (Упр) и S. Список остановился на экране. Для того чтобы продолжить вывод списка файлов, достаточно нажать клавиши Ctrl-Q. Другая возможность — заставить команду выдачи содержимого каталога вывести список файлов в «широком формате», т. е. в несколько столбцов на экране. Для этого достаточно набрать на клавиатуре DIR/W. Результат показан на рис. 1.5. Наконец, если набрать сочетание DIR/P, то вывод списка файлов приостановится в тот момент, когда экран будет полностью заполнен, и для того чтобы его продолжить, достаточно нажать на любую клавишу, например пробел.

Обратим внимание читателя, что в списке файлов существуют незнакомые нам объекты, например имя TEXT, имеющее расширение <DIR>. Этот объект уже не файл, а отдельный каталог файлов, который может содержать другие списки файлов. В этот каталог можно войти, посмотреть файлы в нем и т. п. Но об этом речь пойдет в конце параграфа, пока же читатель может считать, что он работает только в так называем-

```
Volume in drive D has no label
Directory of D:\DOS
```

COUNTRY	SYS	DISPLAY	SYS	XCOPY	EXE	ANSI	SYS	COMMAND	COM
FORMAT	COM	KBDCHSP	COM	DRIVER	SYS	FASTOPEN	EXE	FDISK	COM
NLSFUNC	EXE	PREFOR	EXE	KEYB	COM	KEYBOARD	SYS	MODE	COM
SYS	COM	TIMER	COM	PRINTER	SYS	RAMDRIVE	SYS	SELECT	COM
LCD	CPI	4201	CPI	TIMER	DOC	VDISK	SYS	EGA	CPI
ASSIGN	COM	ATTRIB	EXE	5202	CPI	READ	ME	APPEND	EXE
DEBUG	COM	DISKCOMP	COM	BACKUP	COM	CHKDSK	COM	COMP	COM
FC	EXE	FIND	EXE	DISKCOPY	COM	EDLIN	COM	EXE2BIN	EXE
KEYBDK	COM	KEYBFR	COM	GRAFTABL	COM	GRAPHICS	COM	JOIN	EXE
KEYBSP	COM	KEYBSV	COM	KEYBGR	COM	KEYBIT	COM	KEYBNO	COM
LINK	EXE	MORE	COM	KEYBSW	COM	KEYBUK	COM	LABEL	COM
REPLACE	EXE	RESTORE	COM	PARK	COM	PRINT	COM	RECOVER	COM
SPEED	COM	SUBST	EXE	SETUP	EXE	SHARE	EXE	SORT	EXE
SK	HLP	SK	COM	TREE	COM	DOSEDIT	COM	SKINST	MSG
LEXICON	EXE	TEXT	<DIR>	SKCOMM	COM	SKINST	COM	PHONE	DIR

```
78 File(s)      73728 bytes free
```

```
D:\DOS>
D:\DOS>
D:\DOS>
```

Рис. 1.5. Экран ПЭВМ при выводе списка файлов в широком формате

мом корневом каталоге, в который он попал, перейдя на фиксированный диск.

Просматривая каталог файлов, мы увидели долгожданное сочетание LEXICON . EXE. Редактор действительно оказался записанным на фиксированном диске. Теперь мы можем его запустить, набрав его имя на клавиатуре.

Возникает вопрос: нельзя ли как-нибудь упростить процедуру поиска файлов в каталоге? Действительно, если на фиксированном диске записаны хотя бы 50 файлов, то просматривать весь список в поисках нужного довольно утомительно. Гораздо проще было бы задать компьютеру вопрос типа «Есть ли на диске файл LEXICON?». Этого можно добиться, написав на экране

C)DIR LEXICON . EXE

Если такой файл будет найден, система высветит его на экране.

Но как нам быть, если имя файла известно неточно, например, мы не уверены, что он называется точно LEXICON, может быть, правильно писать имя файла как LEXIKON? Некоторого приближения к командам такого рода можно добиться, используя специальные символы: звездочки и вопросительный знак. Звездочки обычно обозначают что-то вроде «любые символы после этого в имени файла или его расширения». Вопросительный знак обозначает «любой символ вместо этого». Так, дав команду

C)DIR * . EXE

мы получим список только тех файлов, которые имеют расширение . EXE. Дав команду DIR * . * мы, очевидно, повторим просто команду DIR. Теперь мы можем давать команды типа

DIR LEX * . * (найти все файлы, начинающиеся с букв LEX)

DIR L * . EXE (найти все файлы, начинающиеся с буквы L и имеющие расширение . EXE)

В то же время нельзя дать команду DIR LE * N . EXE — система ее просто не поймет.

Команда

DIR LEXI?ON . EXE

означает: найти все файлы с заданным именем, в которых после буквы I стоит любая буква.

Читателю должно быть понятно, что команда

DIR A:

покажет каталог файлов на диске A.

Мы уже научились давать самые разнообразные команды просмотра каталога. Предположим, что мы хотим, чтобы редактор был записан именно на флоппи-диске, так как в скором времени мы будем работать на другой машине, на которой вообще нет фиксированного диска. Для этого нам необходимо

скопировать файл LEXICON . EXE на диск A. Дадим следующую простую команду:

C>COPY LEXICON . EXE A : LEXICON . EXE
которая расшифровывается как «Скопировать файл на диск A: с тем же именем». В русифицированных версиях ДОС эта команда дается как КОП. Команду можно упростить, например ей эквивалентны следующие команды:

COPY LEXICON . EXE A:
COPY LEXICON . EXE A: * . * (скопировать файл с тем же именем)

Читателю должно быть понятно, что команда

COPY LEXICON . EXE A: REDACTOR . EXE
скопирует файл, изменив его имя, т. е. для вызова редактора с диска A: придется набирать REDACTOR . EXE, а команда
COPY * . EXE A: * . *
произведет копирование всех файлов с расширением . EXE на диск A.

Следует обратить внимание читателя на то, что при выполнении операций копирования необходимо следить за имеющимся свободным местом на диске. Свободный объем диска показывает команда DIR, и если размеры копируемого файла превысят имеющееся свободное место, система выдаст сообщение об ошибке «Исчерпано место на диске».

Теперь мы уже умеем копировать файлы. Предположим, что мы захотели стереть наш редактор на фиксированном диске. Для этого дадим команду DEL (русская версия УБР):
DEL LEXICON . EXE

Если мы хотим стереть все файлы, начинающиеся со словосочетания LEX, то для этого достаточно дать команду
DEL LEX * . *. Аналогично для стирания всех текстовых файлов, имеющих расширение . TXT, необходимо дать команду
DEL * . TXT.

Предположим, что мы увидели на фиксированном диске файл со странным названием README . LEX. Что скорее всего может содержаться в этом файле? Название («Прочти меня») явно указывает на то, что в нем записан какой-то информационный текст. Расширение . LEX указывает на то, что этот текст относится к редактору «Лексикон». Дадим следующую простую команду

TYPE README . LEX
(в русской мнемонике ВЫВ). На экране появится текст, содержащий некоторую информацию типа «В этой версии редактора внесены следующие изменения...» и перечисление этих изменений.

Кроме перечисленных простейших функций, у команд ОС имеются дополнительные возможности, которыми, однако, пользоваться приходится довольно редко. Например, содержимое каталога можно выводить не на экран, а в файл или

на принтер. Это делается командой `DIR >list . txt`, где скобка указывает на то, куда будет выводиться список файлов. Другой пример: при копировании можно объединить содержимое двух файлов и записать их в третий. Для этого достаточно дать команду типа

`COPY file1 . txt + file2 . txt file3 . txt`

Теперь, как мы и обещали, рассмотрим возможности создания иерархической файловой структуры. Вначале — что это такое и почему такая структура может быть необходима? Типичная емкость фиксированного диска составляет 10 Мбайт.

Пользователь, работающий на ПЭВМ, как правило решает различные задачи: пишет тексты, компилирует программы, пользуется различными готовыми пакетами программ и т. п. Можно представить, какая путаница возникнет на диске, если все файлы (а их число легко может превысить несколько сотен) будут храниться в одном и том же каталоге на диске. Кроме того, на одной машине могут работать несколько пользователей и их программы и файлы скоро начнут путаться. Для того чтобы упорядочить эту ситуацию, было бы хорошо иметь возможность создать на диске несколько разделов, например в одном месте держать все тексты, в другом — пакеты прикладной графики, в третьем — тексты программ на языке Бейсик и т. п. Если на компьютере работают различные пользователи, хорошо было бы выделить каждому из них свою область на диске и назвать ее по фамилии пользователя, чтобы избежать путаницы.

В операционной системе ДОС есть такая возможность. Можно создать несколько каталогов или директорий, каждый из которых будет иметь свое собственное имя. Раздел самого верхнего уровня обычно называется корневым каталогом, из него можно перейти в любой подкаталог, внутри которого можно создать каталоги более низкого уровня и т. п.

Типичная структура файловой системы такого рода показана на рис. 1.6: в ней имеется корневой каталог `ROOT`, содержащий системные команды ДОС и программы общего назначения. Кроме того, имеется каталог для программ на языке Си и Бейсик и специальный каталог для текстов. Внутри этого каталога произведено разбиение на подкаталоги пользователей Петрова и Иванова, в которых содержатся их файлы, а пользователь Петров кроме этого еще создал специальный каталог `GAMES` (игры), видимо, для того, чтобы поиграть с компьютером в шахматы. Каталоги, 'выходящие из данного, можно увидеть, давая команду `DIR`: они будут выглядеть так же, как обычные файлы, но иметь расширение `<DIR>`. Заметим, что каталог — совершенно отдельный объект, не похожий на файл, его нельзя скопировать или переименовать.

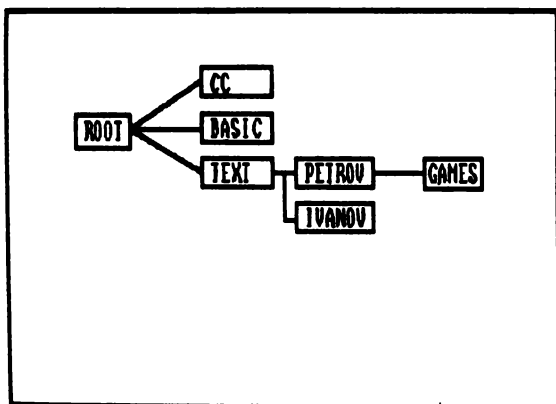


Рис. 1.6. Пример файловой структуры ПЭВМ

Для работы с каталогами в ДОС предусмотрены отдельные команды.

Для того чтобы сделать выбранный подкаталог текущим, необходимо указать системе кратчайший и полный путь к нему по дереву каталогов. Для этого в системе ДОС имеется команда CD (Change Directory) или в русской транскрипции СК. В операционной системе ДОС принято также соглашение о том, что имена каталогов необходимо разделять между собой обратной наклонной линией.

При перемещении по дереву каталогов можно двигаться только «вниз» и нельзя перепрыгивать на другие ветви иным путем, чем через корневой каталог. Это можно легко понять на основе простой житейской аналогии: пусть различные учреждения подчинены одному и тому же министерству. В такой иерархической системе любая бумага, родившаяся в каком-то учреждении, может переходить только в учреждения, непосредственно подчиненные данному, и должна утверждаться «наверху», для того чтобы попасть в другие организации. Чтобы читатель понял, каким образом давать команды на смену текущих каталогов, приведем несколько примеров. Предположим, что мы находимся, как и раньше, в корневом каталоге и мечтаем попасть в раздел BASIC для того, чтобы запустить программу. Для этого достаточно дать команду CD BASIC. Теперь можно забыть о всех остальных каталогах и работать только с текущим списком файлов.

Для того чтобы вернуться в корневой каталог из любого участка дерева каталогов, достаточно дать команду

CD \

Предположим, что из каталога BASIC мы хотели бы оказаться в каталоге текстовых файлов пользователя Петрова. Для этого дадим команду

CD \TEXT\PETROV

Что произошло в результате выполнения этой команды? Мы указали системе полный путь (см. рис. 1.6), который нужно пройти для того, чтобы оказаться в каталоге PETROV: вернуться в корневой каталог (вспомните про министерство!), войти в каталог TEXT и оттуда уже перейти в каталог PETROV. Но если бы мы находились в каталоге TEXT, то для перехода в каталог PETROV (вспомните про мелкую контору!) нам достаточно было бы дать команду

CD PETROV

а для перехода в раздел игр команду

CD PETROV\GAMES

С другой стороны, если мы находимся в каталоге PETROV или в каталоге GAMES, то перейти в каталог IVANOV можно опять-таки только через корневой каталог командой CD \TEXT\IVANOV. Обратим внимание на то, что, указывая путь по дереву каталогов, мы разделяем наклонными чертами только имена каталогов, но никак не имена файлов!

Правила ссылок на каталоги распространяются и на ссылки на имена файлов. Ссылаясь на файл, находящийся в каком-либо другом подкаталоге, необходимо указать полный путь к нему по дереву каталогов. Например, для того чтобы из корневой директории вызвать игру CHESS, находящуюся в каталоге GAMES, нам придется написать

C)\TEXT\PETROV\GAMES\CHESS

а для того чтобы находясь в каталоге IVANOV скопировать на флоппи-диск компилятор языка Бейсик, находящийся в каталоге BASIC, напишем

COPY C:\BASIC\BASCOM.EXE A:*.*

Для создания каталога в ДОС предусмотрена команда MD (Make Directory) или в русской транскрипции НОВКАТ (НК). Для стирания пустого каталога имеется команда RD (Remove Directory) или УБРКАТ (УК).

Итак, мы научились делать восемь из наиболее часто встречающихся операций при работе с ДОС. Перечислим их в условной записи, в которой угловые скобки будут означать, что то, что в них находится, вводить необязательно:

1) (имя диска): — смена текущего диска, например A:, B:, C.;

2) DIR <имя файла> </W> </P> — просмотр каталога, например DIR *.*TXT/W — просмотреть все файлы с расширением .TXT и вывести их в широком формате;

3) COPY (имя исходного файла) (имя конечного файла) — копирование файлов;

4) DEL (имя файла) — стирание файла или группы файлов;

5) TYPE (имя файла) — вывод на экран содержания текстового файла;

6) CD (имя каталога)

7) MD (имя каталога) — создать новый каталог;

8) RD (имя каталога) — стереть пустой каталог.

В первом приближении умения пользоваться этими командами уже вполне достаточно для того, чтобы включить компьютер и запустить на нем готовую программу или разработать свое математическое обеспечение. Приведем еще одну команду (вернее, дополнительную программу ДОС), пользоваться которой необходимо на ПЭВМ, в которых установлен фиксированный диск. Как мы упоминали, фиксированный диск является довольно чувствительным устройством. Для того чтобы избежать повреждения его поверхности, перед любым перемещением компьютера головку диска необходимо зафиксировать с помощью команды PARK (в некоторых версиях ДОС она называется PARKDISK). Эта команда не имеет параметров. После ее выполнения на экране должна появиться надпись типа: «Команда выполнена, головка диска зафиксирована». Теперь компьютер можно выключить из сети и передвигать.

Для минимального закрепления материала рассмотрим следующую задачу. Пусть пользователь Сидоров из дружественной организации принес своему другу Петрову дискету с записанными на ней замечательными программами и не менее замечательными текстами. Сидоров готов разрешить Петрову скопировать программы при условии, что ему будет дана возможность иногда работать на персональном компьютере. Петров согласен на такой вариант. Он решает создать дополнительный каталог SIDOROV, войти в который можно только из каталога PETROV, и скопировать туда принесенные программы. Последовательность команд Петрова должна быть примерно такой:

CD \TEXT\PETROV (вошел в свой каталог из корня)

MD SIDOROV (создал новый каталог SIDOROV)

CD SIDOROV (вошел в этот каталог)

COPY A: * .EXE (скопировал в него программы с гибкого диска)

CD \ (вернулся в корневой каталог)

Но в этот момент Сидоров высказывает недовольство: он хочет, чтобы Петров скопировал также и тексты Сидорова. Тогда Петров вынужден дать следующую команду:

COPY A: * .TXT C:\TEXT\PETROV\SIDOROV* . *

Наконец, рассмотрим более подробно работу с командными файлами, имеющими расширение .BAT. Этот файл можно рассматривать как обычный текстовый файл, в котором записана последовательность команд ДОС. Например, если Сидоров приносит новые программы Петрову довольно часто, то имеет смысл создать файл SIDOROV .BAT, в котором будет записан следующий текст:

CD \TEXT\PETROV\SIDOROV (войти в каталог Сидорова)

COPY A: * . * (скопировать все, что он принес)

Теперь при каждом появлении Сидорова достаточно будет вставить его гибкий диск в компьютер и набрать на клавиатуре SIDOROV, в результате все его файлы окажутся автоматически скопированными.

Мы настоятельно советуем читателю потратить некоторое время на закрепление правил работы с описанными в этом параграфе командами ДОС, а затем просмотреть гл. 3, в которой будут рассмотрены также некоторые другие команды этой операционной системы и работа с надстройками над ДОС.

1.3. СОЗДАНИЕ ТЕКСТА ИЛИ ОПТИМАЛЬНАЯ ПИШУЩАЯ МАШИНКА

По-видимому, практически всем читателям этой книги приходилось печатать на пишущей машинке. Пишущая машинка, конечно, явилась революционным изобретением. Печатать на ней одно удовольствие: из ударов по клавиатуре на бумаге рождается аккуратный текст стандартного размера и формы.

Любому человеку, печатавшему на машинке, известны и такие неперенные аксессуары, как корректирующая жидкость «Штрих» и бумага для исправления ошибок. Вообще, сделать ошибку в процессе печатания текста весьма неприятно: приходится прерываться, выдвигать бумагу, открывать флакончик с корректирующей жидкостью, замазывать опечатку, закрывать флакончик (а то засохнет!), ждать пока паста высохнет, стирать опечатку на других экземплярах под копиркой, вставлять бумагу обратно, пододвигать каретку к месту опечатки и, наконец, победным ударом исправлять допущенную ошибку.

Но вот текст напечатан. Автор придирчивым взглядом смотрит на него и видит, что в тексте имеются повторы, стилистика требует изменения, некоторые абзацы надо переставить местами, в какие-то участки необходимо вставить дополнительные разъяснения и т. д. При этом классической является процедура клея и ножниц: вначале делаются мелкие правки, затем необходимые участки текста вырезаются и меняются местами. Когда исправленный текст готов, его в 95% случаев необходимо перепечатать.

При работе на ПЭВМ создание текстов является одной из наиболее часто встречающихся задач. Во-первых, для того чтобы создать любую программу, ее вначале надо записать в виде текста в специальный файл, а потом многократно редактировать: исправлять ошибки, вставлять дополнительные операторы. Во-вторых, с помощью ПЭВМ можно решить огромное число задач, связанных с подготовкой разнообразных текстовых документов. Для этой цели на ПЭВМ созданы специальные программы, называемые текстовыми редакторами или текстовыми процессорами.

Статистика использования ПЭВМ показывает, что более 70% всех применений ПК связано именно с подготовкой текстов. Этой участи не избежать никому: слишком уже велики удобство и скорость подготовки текста на ПЭВМ. Авторам неоднократно приходилось наблюдать, как после приобретения ПК их счастливые обладатели на несколько месяцев впадали в устойчивый режим подготовки разнообразных текстовых документов: писались статьи, протоколы, служебные записки и т. п. При использовании компьютера для подготовки текстов резко возрастает производительность труда. Во-первых, исчезает страх сделать опечатку: они легко исправляются. Во-вторых, повышается скорость печатания. В-третьих, резко возрастает эффективность переработки текстовых документов.

Работа с редактором текста на ПЭВМ только в одном напоминает работу с пишущей машинкой: перед пользователем находится клавиатура и набранные на ней буквы и числа немедленно отображаются на экране, как будто на листе бумаги. Но все эти символы не напечатаны на нестираемом носителе, а хранятся в памяти компьютера. Это позволяет делать с ними любые операции.

В приведенном примере исправления одной опечатки необходимо было сделать девять последовательных операций, в лучшем случае требующих 40—50 с. Если мы допустили ошибку при печатании слова в текстовом редакторе ПЭВМ, то все, что нам необходимо сделать, это нажать на стрелки передвижения курсора, чтобы установить его на место ошибки, нажать на клавишу «Стирание» и напечатать новую букву или число. Эти три простые операции занимают в худшем случае 10 с, а обычно при исправлении текущих ошибок 2—3 с.

Подготовка текстов с помощью компьютера — действительно революционный скачок по сравнению с пишущими машинками: такой гибкости, скорости и удобства выполнения необходимых операций до сих пор человеку не предоставлялось.

Интересно, что наибольшее развитие редакторы получили именно с появлением ПК: возникла необходимость сделать текстовый редактор максимально удобным и естественным для человека, так как появился огромный рынок для продажи этих программных продуктов. Авторы еще до появления ПЭВМ прошли долгий путь, отчасти повторяющий историю развития редакторов текста на ЭВМ. Вначале их «редактором текста» был стандартный перфоратор. Огромным достижением было, когда перфоратор умел пропечатывать на перфокарте текст набранного кода. Мучения при работе с перфоратором были столь велики, что для нас настоящим потрясением явилась установка дисплеев с так называемым строчным редактором текста. В таком редакторе одновременно можно было редактировать только одну строку текста, т. е. нельзя было передвигать курсор на предыдущую или следующую строку — для этого

было необходимо дать специальную команду. Но и этот строчный редактор текста, который казался нам венцом совершенства, в результате показался гадким уродцем при знакомстве с одним из экранных редакторов. Когда же появился широкий набор экранных редакторов с богатыми возможностями, то нам стало вообще непонятно, каким образом удавалось писать и редактировать тексты и программы без них.

При работе с экранным редактором текста на экране компьютера видна только часть текста — около двадцати строк. При наборе текста последняя строка всегда оказывается свободной, а часть текста, видимая на экране, перемещается вверх, как будто мы печатаем на бесконечной ленте бумаги, проползающей мимо нашего экрана. При этом в любой момент мы можем с помощью клавиш управления курсором передвигаться по тексту, видимому в пределах нашего окна. В том случае, если курсор находится на крайних строчках текущего окна, то очередная команда на перемещение курсора приводит к изменению границ видимой части текста, т. е. текущее окно смещается. Естественным, что на экран можно вывести любой участок этой «ленты», используя специальные клавиши перелистывания страниц.

Итак, мы поняли, что при работе с текстовым редактором мы можем клавишами управления курсором передвигаться по тексту в любом направлении, исправлять ошибки, листать страницы текста. Как правило, в процессе набора текста 70% операций сводится именно к «пролистыванию» и исправлению простых ошибок введения текста. Что еще умеет делать редактор? Вообще говоря, огромное число операций. Прежде всего можно выделить любой фрагмент текста (букву, набор букв, слово, строку, набор строк, абзац и т. п.) и сделать с этим фрагментом разнообразные операции: стереть, скопировать, перенести в другой текст и т. п. Кроме того, можно попросить редактора текста найти символ или сочетание символов в тексте и заменить одно сочетание другим. Можно одновременно редактировать несколько текстов: взять фрагмент из одного и перенести в другой. Можно объяснить программе редактора, что данный текст должен иметь столько-то строк и такой-то абзацный отступ и полностью забыть о необходимости «перевода каретки» при приближении к правому краю страницы. Многие редакторы не только сделают это автоматически, но даже выровняют текст по правому краю и перенесут слова, используя грамматические правила. Можно попросить редактор соблюсти формат какого-либо документа, имеющего заданную форму: официального письма, таблицы и пр. Наконец, можно проверить правильность правописания, вызвав специальную программу!

Если не провести своеобразного «структурирования» функций, выполняемых текстовым редактором, то можно легко утонуть в огромных и толстых руководствах по их использо-

ванию, нередко занимающих несколько томов текста. Часто бывает просто удивительно, что при отточенности и совершенстве программного продукта его описания полны неясностей и имеют огромные размеры. Да и команд у текстовых редакторов может быть несколько десятков. Все они, однако, крайне просты, если исходить из реальных потребностей человека, пишущего текст. Рассмотрим типичный набор команд текстового редактора, упорядоченный по средней частоте их использования:

- считывание и запись текстового файла,
- перемещение курсора по тексту,
- стирание, вставка и замена символов,
- работа с выделенным фрагментом текста,
- поиск в тексте и замена,
- задание формата текстового документа, форматирование,
- задание системных параметров работы редактора,
- проверка правописания.

Рассмотрим эти функции более подробно.

Считывание и запись текстового файла. Прежде всего, для того чтобы начать редактировать текст, необходимо вызвать программу редактора. При этом необходимо указать, в каком файле хранится или будет храниться текст. По окончании работы необходимо дать команду на запись подготовленного текста. Для этого в редакторе обычно предусмотрено специальное меню, в котором есть команды записи и чтения. Для чтения файла, например, может быть необходимо выполнить такую последовательность операций:

- с помощью клавиш движения курсора выбрать в меню раздел «Чтение»,

- нажать «Возврат каретки».

На экране появится приглашение к вводу имени файла или в специальном окне появится список файлов текущего каталога. Теперь необходимо ввести имя файла и нажать на клавишу перевода строки. Теперь файл считан и текст его появляется на экране, а в специальной статусной строке возникает его имя.

В режиме считывания и записи существует ряд дополнительных возможностей. Например, можно «склеить» тексты, записанные в двух различных файлах. Для этого достаточно считать вначале один файл, а затем второй, указав редактору, что он должен разместиться в памяти после первого. Можно считать или записать только некоторый фрагмент текста и т. п.

Перемещение курсора по тексту. Практически во всех редакторах нажатие клавиш управления курсором (вправо, влево, вверх, вниз) смещает текущую позицию в тексте соответственно на один символ или строку в соответствующем направлении. При этом в специальной статусной строке вместе с именем обрабатываемого файла индицируется и текущее положение кур-

сора: номер строки и текущая позиция в строке. Кроме этих простейших операций, клавиши PgUp, PgDn соответствуют переводу страниц: на двадцать строк вверх и вниз. Обычно сочетание этих команд с клавишей Ctrl (Упр) соответствует перемещению на начало или конец всего текста. Таким образом, мы можем перемещаться с помощью клавиш управления курсором в пределах окна, листать страницы и «прыгать» на начало или конец текста.

Другой частой операцией, которую необходимо выполнять при работе с текстом, является расстановка меток. Обычно для этой цели выделяется какая-нибудь функциональная клавиша, например F5. Целью расстановки меток, понятных редактору, но «невидимых» для пользователя, является возможность с помощью одной команды перемещаться на выделенное место текста. Например, по мере работы нам может быть необходимо несколько раз возвращаться к выделенному абзацу, находящемуся где-то в середине текста. Тогда, пометив его, мы можем дать команду «Перейти к метке», и на экране возникнет часть текста, начинающаяся с выделенного участка.

Кроме этих операций движения по тексту, в различных редакторах предусмотрены и некоторые другие. Например, часто необходимо установить текущую позицию в начало или в конец строки. Эта операция обычно делается с помощью клавиш Home, End. Кроме того, курсор можно заставить передвигаться на каждое следующее или предыдущее слово, вставать в начало или в конец абзаца и т. п.

Стирание, вставка и замена символов. Эти команды являются основным механизмом редактирования. Их можно разделить на несколько основных групп. Прежде всего, в ПЭВМ имеется две клавиши для стирания текста: «Забой» и Del. Клавиша «Забой» стирает символ, находящийся слева от курсора, а клавиша Del символ, находящийся непосредственно под курсором. Например, если курсор в слове ОШИПКА установлен под буквой «П» и нажата клавиша «Забой», то в результате на экране окажется написанное слово ОШПКА. В том случае, если нажата клавиша Del, то мы увидим на экране ОШИКА.

Редактор может работать в двух режимах: замены и вставки. В большинстве редакторов эти режимы переключаются нажатием на клавишу Ins. При работе в режиме замены символов мы можем писать новый текст «поверх» старого. Например, пусть слово написано с ошибкой: ОШИПКА. Тогда, установив курсор на букву «П» и набрав вместо нее букву «Б», мы получим правильное написание ОШИБКА. В режиме вставки при этом произойдет следующее: буквы в слове и весь последующий текст раздвинется и в результате на экране будет написано слово ОШИБПКА. Теперь для исправления ошибки необходимо установить курсор под букву «П» и дать команду «Стирание». Режим вставки обладает большой гибкостью: в любое место

текста можно вставлять любое сочетание символов, слов, абзац, новые строки и т. д., при этом весь последующий текст раздвигается. Если в режиме вставки дать команду перевода строки, то в большинстве редакторов текущая строка разорвется на том месте, где стоял курсор. Например, пусть на экране набрана следующая строка:

«Это пример строки, в которой будет произведен разрыв»
Пусть курсор стоит под буквой «т» слова «которой». Тогда при переводе строки в этом режиме произойдет следующее:

«Это пример строки, в ко
торой будет произведен разрыв»

Понятно, что, установив курсор в этом режиме на конец текущей строки и дав несколько последовательных команд «возврат каретки», мы вставим в текст несколько пустых строк. Иногда для такого разрыва и склеивания строк в редакторах предусмотрены специальные команды.

Частой операцией при редактировании текста является также процедура стирания целой строки. Кроме того, предусмотрены более изощренные процедуры стирания: слова, ближайшего слова, находящегося справа или слева от текущего и т. д.

Но как быть, если мы случайно испортили текст, а его копии не успели сделать? В некоторых редакторах для этой цели имеется специальная команда «Откат» или Undo, которая позволяет последовательно отменять операции, которые были выполнены в ходе работы с редактором. Например, если мы случайно стерли кусок текста, то при помощи этой команды мы сможем его восстановить.

Манипулируя описанными простыми процедурами, можно решить практически все мыслимые задачи редактирования текста: исправить ошибки, стереть ненужные куски текста, вставить дополнительные строки. Читатель, который прочел параграф до этого места, может считать, что он уже знает почти все возможности текстового редактора и все, что ему осталось — запомнить конкретную мнемонику этих команд в его редакторе.

Работа с выделенным фрагментом текста. Эта команда используется в тех случаях, когда с выделенным фрагментом необходимо выполнить однотипную операцию. Таких операций по сути не так уж много: копирование, перемещение, стирание, запись на диск. Кроме того, необходимо предусмотреть процедуру выделения необходимого фрагмента. В большинстве редакторов это может быть сделано с помощью двух функциональных клавиш, например F3 — пометить начало фрагмента, F4 — пометить конец фрагмента. При этом выделенный участок текста выделяется цветом. Теперь его можно перенести из данного участка текста в другой, размножить, стереть. Это за-

меняет работу с ножницами и клеем при редактировании текста, напечатанного на пишущей машинке.

Поиск в тексте и замена. Довольно часто возникает необходимость найти в большом тексте какое-то ключевое слово. Это может быть название элемента, переменной или оператора в программе. Для этого в редакторе предусмотрены команды поиска и замены. Предположим, что у нас имеется специальная клавиша «Найти» (например, функциональная клавиша F2). Нажав на нее, мы увидим в верхней части экрана около статусной строки сообщение «Введите, что искать». Предположим, что мы хотим найти в тексте программы переменную INUMBER. Введем ее символьное обозначение и дадим возврат каретки. При этом на экране возникнет часть текста программы, содержащая первую ссылку на эту переменную и курсор будет установлен в первой позиции слова INUMBER. Снова нажав на клавишу «Поиск» или дав специальную команду «Повторить», мы переместимся на второе упоминание этой переменной и т. д. Поиск переменной обеспечивает значительное удобство при редактировании текста или программы. Например, запросив поиск символьного выражения «= INUMBER», мы найдем только те участки программы, в которых значение этой переменной присваивается другим переменным. У программы поиска существует несколько параметров: поиск можно проводить во всем тексте; только в тексте, следующем за или перед текущей позицией курсора, можно потребовать найти заданное число ссылок и т. п.

Аналогичную структуру имеет и команда «Найти и заменить». При этом указывается исходная переменная и контекст замены. Например, мы можем потребовать заменить обозначение INUMBER на NUMBER во всей программе или в нескольких случаях. При этом можно попросить программу замены запрашивать в каждом конкретном случае разрешение на замену: каждая найденная переменная INUMBER в тексте будет окрашена в другой цвет, а в командной строке появится запрос «Заменить?». Если пользователь ответит «Да», то замена будет произведена, в противном случае будет найдена следующая ссылка на переменную и так до тех пор, пока не будет обработан весь текст.

Задание формата и форматирование. Применяется в основном при работе с текстовыми документами. При этом редактору требуется указать ряд параметров, прежде всего: число символов в строке, абзацный отступ, левую границу поля текста, число строк на странице.

Форматирование текста можно произвести уже после того, как текст введен в память ПЭВМ. В большинстве редакторов, однако, имеется возможность установить необходимые параметры перед началом ввода текста. Это обеспечит значительные удобства: не нужно заботиться о переводе строки, так как

в момент, когда текст подходит к правой границе поля, редактор автоматически делает перевод. При этом существует возможность выровнять текст по правому полю. Выравнивание может быть произведено увеличением числа пробелов между словами в строке. Если какие-то слова не помещаются в строке, то предусмотрена возможность их переноса в соответствии с грамматическими правилами. Наконец, существует возможность в любой момент изменить параметры формата и потребовать переформатировать весь текст или его часть.

В текстовых редакторах иногда существует возможность создать довольно сложный формат текстового документа. Например, при подготовке официального письма от организации существует ряд канцелярских требований: адрес должен располагаться в определенном месте листа бумаги, в другом участке листа располагается графа «По вопросу о», в третьем должен начинаться текст и т. д. Такой формат можно создать и запомнить, а при подготовке служебных писем достаточно будет дать редактору команду типа «Использовать формат LETTER. FRM» для того, чтобы избежать ошибок. Кроме этого, можно задать стандартные позиции в строках, к которым должен «прыгать» курсор при нажатии на клавишу табулятора, точно так же, как и на пишущей машинке.

В результате работы форматтера текста получается аккуратный документ, удовлетворяющий самым придирчивым канцелярским вкусам.

Задание системных параметров работы редактора. Пользователь может «настроить» текстовый редактор для своих применений. Под настройкой понимается установка набора системных параметров: цвета и размера текстовых окон, их расположения на экране и количества, начальных параметров инициализации редактора (тип формата документа, имя или типичные расширения обрабатываемых файлов и др.). Например, в большинстве текстовых редакторов используется многооконный режим. Каждое окно может содержать одинаковый или различные тексты и несколько таких текстов могут быть видны на экране одновременно. При этом пользователь может извлекать любые фрагменты из любого окна и переносить их в другое окно. Тогда можно определить стандартный набор окон, видимых одновременно, указать, какие файлы должны быть в них считаны, и при каждом последующем вызове редактора сразу «проваливаться» в необходимое состояние.

Проверка правописания. В последнее время для ПЭВМ появляется все больше специальных программ такого рода, называющихся также спеллерами (speller). Типичный спеллер содержит на дисках словарь из 30—50 тысяч слов и набор правил, по которым он проверяет правильность написания текста, так что недоразумения из-за падежей и окончаний обычно исключены. Если спеллер находит неизвестное ему сло-

во, то обычно он может выдать также список слов, близких к данному по написанию и по звучанию (что особенно актуально для английских текстов). Более того, авторы этой книги пользовались спеллером, выдававшим еще и списки синонимов и предупреждавших, что одно и то же слово повторяется в тексте достаточно часто. Если при проверке правописания слово не найдено в словаре, но пользователь уверен в том, что данное слово написано правильно, то он может ввести его в словарь. Практика показывает, что для человека, работающего в своей конкретной области, после нескольких сеансов работы со спеллером уже не возникает недоразумений, связанных с тем, что объем словаря слишком мал: система «усваивает» лексикон пользователя и быстро оказывается подстроенной под него.

Программы проверки правописания являются незаменимыми для поиска опечаток. Практика показывает, что на 10 страницах текста обычно находится от двух до пяти опечаток, которые трудно обнаружить классическим методом просмотра.

Редакторов для ПЭВМ, совместимых с моделью IBM PC, в настоящее время насчитывается несколько сотен. Наиболее распространенными из них являются, несомненно, WordStar фирмы MicroPro, система SideKick фирмы Borland, Perfect Word, MS-Word, ABC, Brief и множество других. Из встречавшихся авторам спеллеров наиболее распространены Perfect Speller и Turbo Lightning. В СССР наибольшее распространение получил редактор Лексикон (Е. Веселов, ВЦ АН СССР). Из зарубежных систем большинство пользователей ПЭВМ используют, по-видимому, редактор, входящий в систему SideKick или похожие на него версии, такие, как «Бюро» разработки Болгарской АН.

Система «Бюро» удобна тем, что является резидентной, т. е. все время находится в памяти и в любой момент может быть вызвана нажатием одновременно клавиш Ctrl и Alt. Это очень повышает скорость разработки программ: если в процессе компиляции выданы сообщения об ошибках, вместо вызова редактора с указанием имени файла достаточно нажать две клавиши и оказаться в тексте программы. Кроме того, в этой системе встроен ряд удобных сервисных функций: калькулятор, календарь, таблица кодировки символов, имеется развитая система подсказок и объяснения команд и т. д.

Редактор текста «Лексикон» также весьма удобен, содержит подсказки, имеет до 9 различных окон, параметры которых можно легко изменить. На рис. 1.7 показан экран ПЭВМ при работе с этим редактором. Научиться работать с «Лексиконом» можно за несколько часов, тем более что вместе с ним на гибком диске имеется и его описание. Этот редактор является составной частью интегрированной системы «Мастер», которая будет кратко рассмотрена в § 2.9.

Для вызова меню нажмите F10. Для подсказки - F1. Выход из меню - ESC.
Переход в другое окно - клавишу ALT одновременно с цифрой-номером окна

ПОК - g1ava1.3 (490 1,4) АЛТ 25 янв 11:15

1.3 Создание текста или оптимальная пишущая машинка

По-видимому, практически всем читателям этой книги приходилось печатать на пишущей машинке. Пишущая машинка конечно являлась революционным изобретением. Печатать на машинке одно удовольствие: из ударов по клавиатуре на бумаге рождается аккуратный текст стандартного размера и формы.

Любому человеку, печатавшему на машинке, известны такие неперенные аксессуары, как корректирующая жидкость "Штрих" (в народе ее также называют попросту замазкой) и бумага для исправления ошибок. Вообще, сделать ошибку в процессе печатания текста весьма неприятно: приходится прерываться, выдвигать бумагу, открывать флакончик с корректирующей жидкостью, замазывать печатку, закрывать флакончик (а то засохнет!), ждать пока паста высохнет, стирать печатку на других экземплярах под копиркой, вставлять бумагу обратно, пододвигать каретку к месту печатки и наконец победным ударом исправлять допущенную ошибку. Но вот текст напечатан. Автор придирчивым взглядом смотрит на него и видит, что в тексте имеются повторы, стилистика требует изменения, неко-

1=g1ava1.3

a)

Для вызова меню нажмите F10. Для подсказки - F1. Выход из меню - ESC.
Переход в другое окно - клавишу ALT одновременно с цифрой-номером окна

ПОК - g1ava1.3 (490 1,4) АЛТ 25 янв 11:15

1.3 Создание текста или оптимальная пишущая машинка

По-видимому, практически всем читателям этой книги приходилось печатать на пишущей машинке. Пишущая машинка конечно являлась революционным изобретением. Печатать на машинке одно удовольствие: из ударов по клавиатуре на бумаге рождается аккуратный текст стандартного размера и формы.

Любому человеку, печатавшему на машинке, известны такие неперенные аксессуары, как корректирующая жидкость "Штрих" (в народе ее также называют попросту замазкой) и бумага для исправления ошибок. Вообще, сделать ошибку в процессе печатания текста весьма неприятно: приходится прерываться, выдвигать бумагу, открывать флакончик с корректирующей жидкостью, замазывать печатку, закрывать флакончик (а то засохнет!), ждать пока паста высохнет, стирать печатку на других экземплярах под копиркой, вставлять бумагу обратно, пододвигать каретку к месту печатки и наконец победным ударом исправлять допущенную ошибку. Но вот текст напечатан. Автор придирчивым взглядом смотрит на него и видит, что в тексте имеются повторы, стилистика требует изменения, неко-

1=g1ava1.3

б)

Рис. 1.7. Экран ПЭВМ при работе с редактором текста Лексикон в различных режимах:

а) редактирование текста, б) помечен фрагмент текста

Что же необходимо для того, чтобы начать работать с редактором текста, например «Лексиконом»? Опишем кратко последовательность необходимых операций. Пусть пользователь решил создать новый текстовый файл. Для этого он должен вставить диск с записанным на нем редактором в дисковод и написать на экране, на котором горит буква А)

А) LEXICON (возврат каретки)

При этом перед пользователем возникает поле экрана (рис. 1.7, а). Теперь можно писать любой текст, как на пишущей машинке, переводить каретку, двигать курсор, стирать его различные куски. Например, сотрем фрагмент текста, который нас не устраивает. Для этого установим курсор на начало текста и нажмем функциональную клавишу F3. Теперь вся строка окрашена инверсным цветом. Любое передвижение курсора приводит к тому, что следующие строки окрашиваются в такой же цвет. Эта окраска означает, что выделенные строки входят в помечаемую нами область. Пусть мы выделили девять строк (рис. 1.7, б). Теперь нажмем одновременно клавиши Ctrl и F3, и выделенный текст исчезнет с экрана. Такими же простыми командами можно перенести выделенный кусок в другой участок памяти (Ctrl + F4), найти сочетание символов и т. п. Предположим, мы хотим записать текст в файл. Тогда нажмем на функциональную клавишу F10, и в верхней части экрана появится меню. Двигая курсор, выберем функцию «Сохранить», дадим возврат каретки и введем имя файла. Текст записан на диск.

Теперь кратко рассмотрим минимальный набор команд, необходимый для подготовки текста в рамках системы «Бюро».

1. Для вызова редактора необходимо одновременно нажать на клавиши Ctrl-Alt. На экране возникнет главное меню системы. Выберем из этого меню «Редактор» нажатием на функциональную клавишу F2.

2. Для получения подсказки нажмем клавишу F1, при этом на текст будет наложено специальное окно подсказок (рис. 1.8, а). Для выхода из режима «помощь» нажмем на клавишу Escape.

3. Для считывания файла нажимаем клавишу F3. При этом на экране появится окно, содержащее список файлов (рис. 1.8, б). С помощью клавиш управления курсором найдем необходимый файл и дадим возврат каретки.

4. Теперь мы можем начать набирать текст. Стирание и перемещение по тексту производятся стандартным путем: клавишами «Забой» и Del и стрелками управления курсором. Переключение режимов «Замена» и «Вставка» производится клавишей Ins. Для того чтобы курсор прыгнул к началу и к концу строки, используются клавиши Home, End. Для того чтобы стереть текущую строку текста, нужно нажать одновременно клавиши Ctrl-Y.

Для вызова меню нажмите F10. Для подсказки - F1. Выход из меню - ESC.
 Переход в другое окно - клавишу ALT одновременно с цифрой-номером окна

Диск - readme.txt (37 25,1) РЧ 25 янв 17:10

Вставка Абзац РЧ

Смещения курсора	Кнопка РС
На букву влево	←
На строку вверх	↑
На строку вниз	↓
Страница вверх	PgUp
Страница вниз	PgDn

ателям этой книги при-
 Пишущая машинка конеч-
 Печатать на машинке
 атуре на бумаге рожда-
 мера и формы.
 ми. Когда исправленный

Любому человеку, печатавшему на машинке, известны и та-
 кие неперенные аксессуары, как корректирующая жидкость
 "Штрих" (в народе ее также называют попросту замазкой) и бу-
 мага для исправления ошибок. Вообще, сделать ошибку в про-
 цессе печатания текста весьма неприятно: приходится преры-
 ваться, выдвигать бумагу, открывать флакончик с корректиру-
 ющей жидкостью, замазывать опечатку, закрывать флакончик (а
 то засохнет!), ждать пока паста высохнет, стирать опечатку
 на других экземплярах под копиркой, вставлять бумагу обрат-
 но, пододвигать каретку к месту опечатки и наконец победным
 ударом исправлять допущенную ошибку. Но вот текст напечатан.

1==readme.txt

а)

Для вызова меню нажмите F10. Для подсказки - F1. Выход из меню - ESC.
 Переход в другое окно - клавишу ALT одновременно с цифрой-номером окна

Диск ЗНН - readme.txt (37 10,14) РА 25 янв 17:16

Вставка Абзац РЧ

Список файлов на диске:	
README.TXT	GLAVA2.TXT
GLAVA3.TXT	README.DOC
ARTICLE.TXT	GLAVA4.DOC
GLAVA5.DOC	ANNOT.TXT

ателям этой книги при-
 Пишущая машинка конеч-
 Печатать на машинке
 атуре на бумаге рожда-
 мера и формы.
 ми. Когда исправленный

Любому человеку, печатавшему на машинке, известны и та-
 кие неперенные аксессуары, как корректирующая жидкость
 "Штрих" (в народе ее также называют попросту замазкой) и бу-
 мага для исправления ошибок. Вообще, сделать ошибку в про-
 цессе печатания текста весьма неприятно: приходится преры-
 ваться, выдвигать бумагу, открывать флакончик с корректиру-
 ющей жидкостью, замазывать опечатку, закрывать флакончик (а
 то засохнет!), ждать пока паста высохнет, стирать опечатку
 на других экземплярах под копиркой, вставлять бумагу обрат-
 но, пододвигать каретку к месту опечатки и наконец победным
 ударом исправлять допущенную ошибку. Но вот текст напечатан.

1==readme.txt

б)

Рис. 1.8. Работа с редактором текста «Бюро» в различных режимах
 а) вызов окна подсказки; б) выбор рабочего файла из списка;
 в) вид экрана в режиме редактирования текста

5. Для выделения блока используются функциональные клавиши F7 (начало блока) и F8 (конец блока). Теперь, нажав на клавиши Ctrl-K, мы можем скопировать выделенный блок (дополнительно нажмем на C, стереть его (нажимая на Y) или передвинуть блок (клавиша V).

6. Для поиска сочетания букв в тексте нажмем одновременно клавиши Ctrl-Q, а затем F. Для поиска и замены вместо F нажмем на клавишу A.

7. Для записи файла на диск нажмем функциональную клавишу F2.

Мы рассмотрели почти все необходимые функции редактора. Для работы с ним запомнить нужно только пять-шесть сочетаний клавиш, да и то их мнемоника в любой момент может быть получена путем нажатия на клавишу «Подсказка».

Работать с редактором, как видим, ненамного сложнее, чем с обычной пишущей машинкой, но зато во много раз удобнее. Итак, текстовый редактор является мощным и эффективным средством, заменяющим пишущую машинку. По-видимому, программы редактирования текстов сейчас достигли уже такого уровня, что многие из них с полным правом можно назвать идеальной пишущей машинкой (следующий шаг будет сделан только тогда, когда ПЭВМ начнет выполнять роль стенографистки).

1.4. КАК ЗАСТАВИТЬ ПЭВМ ВЫПОЛНЯТЬ ПРОГРАММЫ И КАК НАУЧИТЬСЯ С НИМИ РАБОТАТЬ

В предыдущих параграфах мы уже познакомились с основными командами ДОС и с принципами работы с текстовым редактором. Основная цель данного параграфа — дать читателю общее понятие о том, что же такое программа с точки зрения компьютера и каким образом запустить ее на ПЭВМ.

Поговорим о том, как запустить стандартные программы: прикладные системы, специализированные пакеты программ и т. п. Если «готовая» программа уже записана в ПЭВМ в виде файла с расширением .EXE, то в большинстве случаев для ее запуска, как мы уже упоминали, необходимо просто набрать имя этой программы. При этом компьютер найдет на диске файл с введенным именем, содержащий подготовленные к исполнению машинные коды, загрузит его в память и начнет выполнение инструкций.

В ряде случаев ситуация сложнее. Например, некоторые программы требуют для своей работы предварительного запуска некоторых других так называемых «резидентных» программ. При вызове резидентных программ не происходит никаких видимых изменений на экране: они лишь заполняют

часть оперативной памяти кодами, необходимыми для работы основной программы. В таких случаях прикладная программа обычно вызывается с помощью специального командного файла, имеющего расширение .BAT. Например, рассмотрим следующую вполне жизненную ситуацию. Пользователь получил оригинальную программу, написанную его другом, и мечтает поскорее запустить ее. Просматривая каталог файлов на гибком диске, на котором записана эта программа, он видит следующие файлы:

DRIVER. COM
PROGRAM. EXE
RUN. BAT

Если попытаться запустить программу, набрав на клавиатуре PROGRAM, то, скорее всего, ничего не получится: программа вскоре «зависнет». Правильная последовательность запуска программы — вначале набрать на клавиатуре DRIVER, дать возврат каретки, а затем уже набрать имя программы. Именно эта последовательность и записана в файле RUN. BAT. Прочтя этот файл текстовым редактором, можно увидеть, что он состоит из двух строк:

DRIVER
PROGRAM

Следовательно, для запуска программы достаточно набрать на клавиатуре RUN и дать возврат каретки. Таким образом, пытаясь запустить новую программу, необходимо всегда попытаться вначале проанализировать возможные варианты: просмотреть все файлы с расширением .BAT, установить, имеются ли файлы с расширением .COM, и только потом начинать загрузку.

При перенесении на ПЭВМ новых программ необходимо иметь в виду, что в большинстве случаев эти программы необходимо «настроить» на конкретную периферию ПЭВМ. Разнообразие периферийных устройств ПЭВМ весьма велико, на зарубежных машинах используется десяток типов графических адаптеров, мониторов, принтеров, графопостроителей и т. п. Обычно вместе с такими прикладными программами, записанными на гибких дисках, поставляются и специальные программы для их «установки». Они могут называться так:

INSTALL (.BAT или .EXE)
SETUP (.BAT или .EXE)

При вызове такой программы или командного файла обычно на экране появляются подробные инструкции типа: «Вставьте диск номер 1 в дисковод, нажмите на возврат каретки», «Укажите, имеется ли в вашей машине математический сопроцессор», «Выберите тип вашего монитора: 1 — черно-белый, 2 — цветной стандартный, 3 — цветной улучшенной графики» и т. д. После того, как пользователь пройдет эту последовательность, на диске образуется файл с записанными параметрами

конфигурации данной ПЭВМ и при последующих вызовах основной программы параметры будут задаваться исходя из содержимого этого файла.

При выполнении программы пользователя на ПК необходимо выполнить следующие операции:

- подготовить исходный текст программы на одном из языков программирования;

- вызвать программу-компилятор, преобразующую текст в последовательность кодов процессора;

- вызвать программу сборки, которая находит все недостающие ссылки на внешние процедуры и подготавливает выполнимый файл;

- вызвать программу.

При создании программ эту стандартную последовательность операций приходится выполнять многократно: находить ошибки, модифицировать исходный текст, снова компилировать и собирать программу, запускать, убедиться, что она работает неправильно и т. д.

Принципы подготовки текста программы мы уже рассмотрели. Теперь поясним необходимость компиляции программы. Что же такое компилятор (или транслятор)?

Процессор компьютера умеет работать с определенным набором очень простых машинных инструкций: записать число в регистр, считать ячейку памяти, прибавить к содержимому регистра число и т. п. Каждая машинная инструкция имеет свой определенный код. В принципе, если выучить эти коды, то любую программу можно записать, используя доступные машинные инструкции. Когда-то программирование ЭВМ велось именно таким образом. Этот процесс можно значительно упростить. Во-первых, каждому из кодов можно присвоить определенную мнемонику, запомнить которую человеку значительно проще, чем сочетания цифр. Тогда программа может записываться с использованием определенных мнемонических операторов, а специальная программа будет переводить эти операторы в машинные коды. Такой «язык программирования» называется также ассемблером. Во-вторых, созданы языки высокого уровня, на которых запись программ приближена к более естественным для человека правилам. Для того чтобы программы, написанные на таких языках, могли быть выполнены, на ЭВМ применяются специальные программы-компиляторы, выполняющие перевод команд такого языка в машинные коды.

Из описанной последовательности действий бывает исключение: на ПЭВМ широко распространены так называемые интерпретирующие языки, чаще всего Бейсик. Интерпретирующий язык подготавливает готовую к выполнению программу в процессе ее выполнения: каждый следующий оператор программы переводится в машинный код сразу только тогда,

когда до него дошла очередь. Понятно, что выполнение программы в интерпретирующем режиме в несколько раз медленнее, чем при использовании компиляторов. В таком интерпретирующем языке все стадии разработки программы обычно совмещены в одной: пользователь может сразу же редактировать текст программы, запускать ее, прерывать исполнение. Работа с интерпретирующим языком годится для обучения и прогонки небольших простых программ. Любую сколь угодно серьезную программу неизбежно придется писать ориентируясь на использование компилятора. В конце этого параграфа мы приведем пример работы с интерпретатором Бейсика.

Рассмотрим последнюю стадию — сборку (link). Необходимость этой операции вызвана несколькими обстоятельствами. Во-первых, при использовании языков высокого уровня часть стандартных функций (операции ввода-вывода, математические функции) записана в стандартных библиотеках. Это означает, что компилятор просто вставляет в последовательность машинных кодов ссылки на соответствующие процедуры, находящиеся в других файлах. Естественно, что при подготовке машинного кода к выполнению необходимо «связать» коды исходной программы с кодами стандартных функций. Во-вторых, программа пользователя может состоять из нескольких программ, коды которых записаны в различных файлах и при подготовке программы к исполнению необходимо каким-то образом указать, в каких именно файлах содержатся необходимые программы, и объединить машинные коды.

Хотелось бы отметить, что компиляторы и сборщики являются весьма сложными программами, содержащими множество различных параметров, которыми пользователь может управлять, добиваясь самых разнообразных эффектов. Так, при компиляции программы можно задать десятки условий, например проводить оптимизацию по размеру кода или по быстродействию и т. п. Но все же в большинстве случаев начинающему пользователю для выполнения компиляции и сборки достаточно знать две стандартные команды.

Приведем простой пример, демонстрирующий последовательность разработки программы на ПЭВМ. Пусть читателя не пугает, что эта программа будет написана на языке Си: она настолько проста, что будет понятна всем.

Войдем в текстовый редактор и наберем следующий текст:

```
# include <stdio.h> /* Включаем ссылку на стандартную библиотеку ввода-вывода */
main ( ).           /* Начало главной программы */
{
short i;            /* Переменная i — целое число */
for (i=0; i <= 10; ++i) /* Делать от i=1 до i=10 */
{
```



```
printf ("\\n Число=% d", i); / * Написать на экране  
Число=i */
```

```
/ * Конец главной программы */
```

Мы написали программу, которая выведет на экран десять чисел от единицы до десяти. Дадим команду на запись файла, назовем его, например, TEST. С. Выйдем из редактора. Любопытства ради можно проверить, действительно ли редактор создал файл TEST. С. Для этого дадим команду DIR TEST. * и убедимся в том, что этот файл действительно создан. Теперь нам необходимо вызвать компилятор языка Си. Предположим, что он загружен в нашу машину и называется MSC. Для компиляции файла наберем простую команду

```
C>MSC TEST;
```

На диске мигает лампочка и через несколько секунд снова появится надпись типа «Компиляция завершена, ошибок не обнаружено» и приглашение операционной системы. Файл скомпилирован! Если бы мы допустили синтаксические ошибки в программе, то на экране появились бы сообщения об ошибках с указанием того, в какой строке ошибка обнаружена и в чем она заключается, например «Пропущена закрывающая скобка» или «Неправильное использование типов переменных» и др. В этом случае нам бы пришлось вернуться в текстовый редактор, исправить ошибку и опять начать компиляцию. Если мы снова посмотрим содержимое каталога на диске, то увидим, что теперь у нас создано уже два файла с именем TEST: TEST. С и TEST. OBJ. Файл TEST. OBJ является результатом деятельности компилятора и содержит код нашей программы.

Проведем операцию сборки. Для этого в нашем случае достаточно дать команду

```
LINK TEST;
```

Опять начнет мигать лампочка, и через несколько секунд появится приглашение операционной системы. Посмотрев на каталог, мы увидим, что у нас появился третий файл: TEST. EXE — это собственно код программы, готовой к исполнению. Для того чтобы запустить программу, просто наберем на клавиатуре TEST и нажмем возврат каретки. На экране побежит надпись:

```
Число=1
```

```
Число=2
```

```
...
```

Разработка любой самой сложной программы состоит из тех же элементарных шагов: редактирования, компиляции, сборки и запуска.

Мы надеемся, что теперь читатель представляет, как выглядит процесс разработки программы.

Программа типа той, которая была приведена в качестве тестового примера, не содержит никаких входных параметров: она методично пишет на экране слова и числа. Эта ситуация с полным правом может быть названа исключительной: 99% всех прикладных программ основаны на диалоге с пользователем. При вызове программ на экране возникают различные подсказки, меню и т. п. Вообще, организация диалога с пользователем — важнейшая составная часть программного обеспечения ПЭВМ. Каковы наиболее типичные ситуации, с которыми пользователю придется столкнуться при работе с различными прикладными системами?

В простейшем случае по мере выполнения программы пользователь должен вводить различные числа или символы в ответ на появляющиеся на экране вопросы. Типичные примеры такого рода:

«Введите значение»

При этом пользователь должен набрать на клавиатуре число и дать возврат каретки. Другой пример:

«Продолжить выполнение программы или выйти?»

Опытный пользователь сразу поймет, что для продолжения ему необходимо нажать на «П», а для выхода из программы — на «В». Чаше встречаются запросы более понятной формы типа:

«Продолжить выполнение программы? (да/нет)?» или

«Продолжить выполнение программы (1/0)?»

Процедура ответов на вопросы может быть организована различным способом: иногда может быть достаточно только нажать на букву «д» (или «1») и программа продолжит работу, иногда необходимо набрать «да», а иногда (чаще всего) необходимо не только набрать «да», но и дать возврат каретки.

В прикладных пакетах программ наиболее распространен метод организации диалога с помощью набора меню. Обычно такие меню возникают в специальных выделенных окнах экрана. Методы работы с меню достаточно похожи друг на друга, но мы все же считаем необходимым рассмотреть их более подробно.

Простейшее меню: на экране высвечен ряд разделов программы, обозначаемый буквами или цифрами, например

1. Построение графика
2. Задание параметров экрана
3. Выход

Введите Ваш выбор:

Для выбора пункта программы необходимо ввести соответствующую букву или цифру и дать возврат каретки. Такой способ, однако, встречается довольно редко. Чаше в диалоговых программах переключение пунктов меню осуществляется с помощью клавиш управления курсором или при помощи «мыши». Нажимая на стрелки управления курсором, мы передвигаемся

по пунктам меню, при этом текущий пункт как правило выделяется перемещающейся рамкой или цветом. Для выполнения выбранного пункта обычно необходимо дать возврат каретки, однако в некоторых программах для той же цели служит пробел.

Если меню достаточно длинное, то передвигаться каждый раз по нему с помощью клавиш управления курсором и нажимать на возврат каретки может оказаться неудобным. Поэтому помимо вызова с помощью движения курсором по пунктам меню в большинстве пакетов каждый из этих пунктов можно немедленно вызвать с помощью нажатия на некоторую стандартную клавишу. Это могут быть функциональные клавиши, первая буква соответствующей строки, единственная заглавная буква в строке, буква, помеченная цветом. Например, в следующем меню

Графика

Графический вывод

Выход

первый раздел вызывается клавишей «Г», второй — клавишей «Р», а третий — клавишей «В».

Как правило, пунктам меню может соответствовать не только немедленно исполняемая команда, но и отдельное меню более низкого уровня. При работе с такими иерархическими системами меню существует множество возможностей.

Во-первых, каждое следующее меню может просто заменить собой предыдущее, накладываясь на его изображение на экране. При этом в появившемся списке обязательно присутствует пункт «Выход» или «Предыдущее меню». Если таких уровней меню достаточно много, то возвращаться в исходное меню, последовательно выбирая предыдущие разделы, неудобно. В этом случае в списке присутствует пункт «Основное меню». Во-вторых, меню более низкого уровня могут возникать в отдельных окнах экрана, не совпадающих с исходным. Это может оказаться полезным для того, чтобы пользователь все время видел перед собой тот «путь», который он прошел по системе меню.

Расположение различных окон в меню может быть разнообразным: они могут быть собраны в определенной части экрана, расположены в верхней его части в виде одной строки, накладываясь друг на друга со сдвигом и т. п. Некоторые примеры использования меню в диалоговых пакетах читатель может найти в рисунках гл. 2.

Кроме того, во многих пакетах программ имеется также возможность введения текстовых команд в специальной командной строке. Обычно эти команды дублируют некоторые разделы меню. Такой командный интерфейс удобен, так как, оставаясь в рамках одного и того же раздела меню, можно выполнять команды в «далеких» от текущего разделах.

Разберем простейший пример диалога на Бейсике. Конечно, Бейсик — это далеко не лучший язык для работы с ПЭВМ, но для изучения клавиатуры и программирования простейших функций это то, что надо. Откроем дисковод и вставим в него дискету с интерпретатором Бейсика. Закроем дисковод и напишем на клавиатуре BASIC, дав возврат каретки. Компьютер обратится к диску, экран мигнет, на нем появится обозначение версии Бейсика и строка OK. Обратим внимание на то, что снизу экрана появится меню, как это показано на рис. 1.9.

```
GW-Basic 3.20
(C) Copyright Microsoft 1983,1984,1985,1986
60332 Bytes free
OK
—
```

```
1:LIST 2:RUN 3:LOAD" 4:SAVE 5:CONT 6:,"LPT1 7:TRON 8:TROFF 9:KEY 10:SCREEN
```

Рис. 1.9. Экран ПЭВМ при работе с интерпретатором Бейсика

Основные функции, которые будут нам нужны из этого меню: функциональная клавиша F1 выдает текст программы, клавиша F2 начинает ее исполнение. Теперь мы находимся в интерпретаторе Бейсика. Написать нужную нам программу можно как на обычной пишущей машинке, вводя текст и переводя строки нажатием на клавишу возврата каретки. Текущие опечатки можно исправлять, нажимая на клавишу «Забой». Если мы допустили ошибки в предыдущих строчках, можно с помощью клавиш управления курсором передвинуться по тексту вверх, исправить найденную опечатку и дать возврат каретки.

В древнем персидском стихотворении сказано:

Тот, кто не знает и не знает, что он не знает, — глупец,
избегай его.

Тот, кто не знает и знает, что он не знает, может научиться,
научи его.

Тот, кто знает и не знает, что он знает,
спит, разбуди его.

Тот, кто знает и знает, что он знает,
— пророк, учись у него.

Поэт так никогда и не узнал, что он мог выразить то же самое так:

C1 Он знает 0 0 1 1

C2 Он знает C1 0 1 0 1

Избегай его X

Научи его X

разбуди его X

Учись у него X

Соответствующая программа на Бейсике выглядит следующим образом:

```
5 REM PERSIAN VERSE
10 PRINT «Персидская версия экспертной системы (Э. Хам-
    би,»
15 PRINT «Программирование таблиц решений, Мир,
    1976 г.)»
20 PRINT «Введите, знает ли ОН? (1 — да, 0 — нет)»;
30 INPUT C1
40 PRINT «Введите, знает ли ОН, что он знает? (1 — да,
    0 — нет)».
50 INPUT C2
60 IF C1=0 AND C2=0 THEN PRINT «Избегай его»
70 IF C1=0 AND C2=1 THEN PRINT «Научи его»
80 IF C1=1 AND C2=0 THEN PRINT «Разбуди его»
90 IF C1=1 AND C2=1 THEN PRINT «Учись у него»
100 PRINT «Если хотите еще набраться мудрости, скажите
    YES»
110 INPUT C3$
120 IF C3$=«YES» THEN GOTO 20
130 PRINT «Желаем успеха в чтении книги!»
140 PRINT «авторы»
150 END
```

В этой программе пользователь должен ввести цифру 1, если ответ на вопрос положительный, и 0, если ответ отрицательный. Программа проверит логические условия и напечатает на экране ответ. Попробуйте набрать этот текст. Нажмите на клавишу F2 (запуск программы). Теперь вы можете отвечать на вопросы и получать ответы системы, основанные на рекомендациях древнего персидского поэта. Для выхода обратно в операционную систему после окончания работы с программой наберите на клавиатуре system и дайте возврат каретки.

Мы надеемся, что, прочтя эту главу, читатель получил представление об основных принципах работы с ПК. Теперь он представляет себе

структуру ПК и его периферийные устройства;

простейшие команды ДОС: умеет находить, стирать и копировать файлы;

принципы работы с текстовым редактором;

типичный процесс разработки программы и использования стандартных пакетов прикладных программ.

Короче говоря, читатель, осиливший первую главу, получил необходимые знания для того, чтобы писать на ПК тексты и запускать программы. Обзор существующего программного обеспечения ПЭВМ сделан в следующей главе.

Глава 2. ИНЖЕНЕРНО-ТЕХНИЧЕСКИЕ ПРИМЕНЕНИЯ ПЕРСОНАЛЬНОЙ ЭВМ: ПАКЕТЫ ПРОГРАММ И РАБОТА С НИМИ

В этой главе мы попытаемся сделать обзор математического обеспечения ПК, правда, довольно поверхностно. Мы почти не описываем работу с конкретными пакетами — сейчас их насчитывается несколько тысяч. Поэтому всюду в этой главе читатель будет знакомиться с некоторым «усредненным» пакетом программ и с типичной структурой команд меню. Это относится и к рисункам: основная их цель дать читателю понятие о наиболее характерной структуре диалога и о представлении данных в прикладных системах ПЭВМ.

Прочтя эту главу, читатель, по-видимому, начнет представлять себе исключительное богатство и разнообразие программного обеспечения ПЭВМ. Фактически можно считать, что компьютер в этой ситуации является лишь прибором для чтения, а где-то на полках стоят стеллажи с дисками, на которых записан коллективный труд сотен тысяч программистов и ученых — практически объем крупной библиотеки с фондом в десятки тысяч томов. Для того чтобы освоить любую из рассмотренных нами прикладных систем, потребуется провести за экраном от нескольких дней до нескольких недель или даже месяцев, да еще и прочесть толстые тома описаний. Этого процесса не надо бояться: принципы организации диалога с ПЭВМ в настоящее время достаточно однотипны, поэтому сложность здесь заключается не столько в общении с компьютером, сколько в понимании предметной области: что именно и как мы хотим сделать. Следует также отметить, что стоимость программного обеспечения, если речь идет о крупных системах типа систем автоматизированного проектирования, нередко превышает стоимость ПЭВМ.

Читатель должен достаточно четко представлять, что прикладное математическое обеспечение не появится у него само собой: в 80% случаев необходимые прикладные пакеты придется длительное время искать, выбирать, закупать. Мы советуем не пожалеть на это времени и попытаться установить на своей ПЭВМ минимальный комплект необходимых прикладных пакетов программ — это значительно облегчит дальнейшую работу. Даже если приобретенные прикладные системы в итоге окажутся малоудобными и не будут удовлетворять пользователя, работа с ними наверняка подскажет ряд решений, которые пользователь затем сможет реализовать в своих программах. В противном случае существует реальная опасность изобретения «велосипедов» и значительных потерь рабочего времени.

2.1. АВТОМАТИЗАЦИЯ И УПРАВЛЕНИЕ ПРОЦЕССАМИ

Компьютер, не связанный с внешним миром ничем, кроме клавиатуры, с которой пользователь вводит программы и числовые данные, может применяться для решения лишь математических, конструкторских и расчетных задач. А такие задачи в целом составляют не более половины различных инженерных приложений. Другая половина технических задач непосредственно связана с применением ПЭВМ в качестве гибкого устройства управления оборудованием и в качестве устройства для контроля состояния аппаратуры и исследования каких-либо процессов. При этом ПЭВМ становится неотъемлемой частью прибора, станка, управляющей системы и т. п.

Любой читатель, прошедший обучение в техническом вузе, помнит многочисленные лабораторные работы, посвященные измерениям различных зависимостей: необходимо длительное время снимать показания, перенастраивать приборы, записывать данные, а затем строить по ним график. Этот процесс занимает обычно длительное время, не говоря уже о множестве ошибок, вносимых как несовершенством процесса, так и из-за ослабления внимания.

В связи с усложнением технических задач возникает все больше случаев, в которых использование человека для управления техническими установками и регистрации данных просто невозможно из-за объема, скорости и сложности выполняемых операций. Рассмотрим два простейших примера.

Задача 1. Пусть на экране ПЭВМ подготовлено изображение разводки простейшей электронной схемы и заданы координаты точек платы, в которых необходимо просверлить отверстия: (X_1, Y_1) , (X_2, Y_2) и т. д. Пусть у нас имеется сверлильный станок с двумя двигателями, которые смещают положение сверла по координатам X и Y , причем смещение пропорционально напряжению, поданному на вход устройства управления двигателями. Пусть у станка имеется специальный цифровой выход, на котором появляется условная комбинация о том, что перемещение сверла закончено. Имеется также возможность начать сверление при подаче на вход устройства управления станком специального цифрового кода. При этом известно, что сверление должно продолжаться некоторое заданное время T , после чего его можно прекратить, подав другой цифровой код на вход устройства управления станком.

Понятно, что для автоматизированной подготовки макета электронной платы нам надо решить сразу несколько задач: научиться подавать на два независимых аналоговых входа станка меняющиеся напряжения, задаваемые программным путем:

научиться подавать и считывать с цифровых входов станка программно-задаваемые цифровые коды;

научиться отсчитывать определенное время.

Задача 2. Для решения задачи оперативного выборочного контроля выпускаемых приборов необходимо создать автоматизированную систему для снятия семейства передаточных функций прибора по напряжению в зависимости от некоторого управляющего напряжения (типичный пример — регистрация вольт-амперных характеристик). Необходимо также определить, укладываются ли найденные параметры в технологический разброс. Для этого необходимо:

подавать на различные входы испытательной колодки изменяющиеся в заданных пределах напряжения;

считывать с выхода получаемые значения напряжения и преобразовывать их в цифровую форму;

по получаемым данным строить на экране семейство графиков и рассчитывать необходимые параметры (точки перегиба, крутизну ВАХ и т. п.).

Понятно, что рассмотренные задачи примитивны и в реальной жизни приходится иметь дело с гораздо более сложными ситуациями: считывать большие массивы данных по многим каналам, производить расчеты на основании достаточно сложных математических моделей, на их основе выдавать комплексы управляющих сигналов на различные внешние устройства и т. д. Короче говоря, общая схема применения ПЭВМ в устройствах управления и автоматизации включает в себя функции считывания информации о состоянии исследуемого объекта по многим каналам, функции модельной модификации этого состояния и функции выдачи управляемых сигналов на внешние исполнительные устройства. Любой читатель легко придумает разнообразные задачи такого рода, которые можно решать с помощью ПЭВМ.

Практически все мыслимые задачи, связанные с автоматизацией и управлением процессами, основаны на очень небольшом наборе необходимых операций и практически все они уже встретились нам при рассмотрении двух простых примеров. Перечислим эти операции:

1. Считывание с одного или нескольких входных источников аналоговых сигналов в форме напряжения и преобразование этих сигналов в цифровую форму (аналого-цифровое преобразование).

2. Преобразование цифровых значений в аналоговый сигнал (напряжение) и выдача этого сигнала на внешние устройства по нескольким каналам (цифро-аналоговое преобразование).

3. Считывание с внешних стандартных цифровых входов цифровой информации (цифровой ввод).

4. Посылка цифровых кодов на внешние устройства (цифровой вывод).

5. Точный отсчет временных интервалов в одном или нескольких независимых устройствах (таймирование).

Понятно, что для реализации перечисленных операций необходимо использовать специализированные устройства. Действительно, если функции цифрового ввода или вывода на ПЭВМ еще можно реализовать через стандартные порты ввода-вывода, а простейшие функции отсчета интервалов можно выполнять с помощью встроенных в компьютер часов, то для ввода и вывода аналоговой информации необходимы специализированные устройства. Развитие необходимых периферийных устройств в настоящее время идет по следующим направлениям.

Одним из них является создание вынесенной из ПЭВМ унифицированной аппаратуры для ввода и вывода сигналов. К такой аппаратуре можно отнести прежде всего КАМАК, разработанный еще в 60-х годах для автоматизации исследований в ядерной физике. В 70-е годы КАМАК являлся одним из наиболее распространенных международных стандартов подобной аппаратуры. В чем заключается привлекательность и основной принцип системы КАМАК? КАМАК представляет собой стандартную корзину (крейт), имеющую 24 позиции, в которые можно вставлять унифицированные модули. Число таких модулей составляет несколько сотен наименований: цифро-аналоговые и аналого-цифровые преобразователи, цифровые регистры (для ввода и вывода цифровой информации), мультиплексоры, служащие для подключения к одному источнику нескольких внешних устройств, модули дополнительной памяти, программируемые усилители, устройства управления шаговыми двигателями, счетчики событий и многие другие модули.

Имея такое большое разнообразие специализированных устройств, можно, как из конструктора, собрать в крейте автоматизированную систему почти на все случаи жизни. Для соединения крейта КАМАК с ЭВМ используется специальный контроллер, выпускаемый для каждого типа вычислительной машины. Существует также возможность собрать систему из нескольких модулей КАМАК. Наиболее разветвленные системы автоматизации на основе КАМАКа используются в ядерной физике и включают до сотен крейтов, причем в крейтах часто устанавливаются специальные микропроцессоры для автономного сбора данных. Программирование системы КАМАК осуществляется на основе следующих принципов: у каждого крейта и у каждой позиции в крейте имеется номер, имеются специальные линии данных и коды функций. Например, пусть в математическом обеспечении ПЭВМ имеется подпрограмма CAMAC (NCRATE, NSTATION, CODE, FUNCTION, DATA)

где параметры подпрограммы, соответственно, номер крейта, номер позиции в крейте, код выполняемой операции, тип операции (запись или чтение) и собственно число, получаемое в результате данной операции. Пусть в крейте 1 в позиции 10 уста-

новлен модуль АЦП, который производит считывание данных по коду 16, причем в разделе FUNCTION при этом должен быть установлен параметр 0. Тогда вызовом подпрограммы

CALL CAMAC (1, 10, 16, 0, IDATA)

мы произведем считывание значения напряжения, подаваемого на этот АЦП, и поместим полученное число в переменную IDATA. Необходимо учесть, что у каждого выпускаемого модуля КАМАК имеются свои коды управления и в общем случае процесс написания программы для автоматизации эксперимента довольно трудоемок. Существуют специальные языки для программирования системы КАМАК, наиболее распространен из них диалект Бейсика, который называется Квейсик. Цикл данных в КАМАКе занимает время 1 мкс, следовательно, предельная частота считывания данных может достигать 1 МГц. Однако практика показывает, что при использовании машины класса ПК и программировании на языке ассемблера предельная частота считывания данных редко превышает 50 КГц, а при использовании языка высокого уровня типа Квейсика она уменьшается до 2 КГц.

Таким образом, система КАМАК имеет гибкую структуру, универсальна, имеет интерфейсы к самым различным типам ПЭВМ. Но одновременно с этим она имеет и значительные недостатки: большие габаритные размеры, высокую цену основных компонентов, относительно низкую надежность. Кроме того, существуют значительные трудности с выпуском и приобретением различных стандартных модулей системы. Стандарт КАМАК был оправдан в период широкого распространения больших и мини-ЭВМ, но в настоящее время из-за перечисленных недостатков от него начинают отказываться. В то же время модульный принцип построения системы оказался весьма удачным, и в настоящее время фирмы начали выпуск специализированных систем сбора данных и управления процессами, отдаленно напоминающих КАМАК, но имеющих меньшие стоимость, размеры и более высокие технические характеристики. Примером такой системы может служить модульный набор LabLink. Для ПК также выпускается широкий набор различных измерительных устройств на базе интерфейса IEEE 488. При этом в компьютер вставляется специальная интерфейсная плата, к которой подключаются стандартные внешние устройства. Но, несмотря на свои недостатки, система КАМАК, видимо, еще не одно десятилетие сохранится в качестве инструмента для автоматизации и управления процессами.

С появлением ПЭВМ все большее распространение получает другой подход, связанный с разработкой специализированных устройств для ввода и вывода данных непосредственно на шину ЭВМ без промежуточных устройств. Этот подход имеет ряд важных преимуществ: повышение эффективности работы системы, большая простота программирования, ком-

пактность и дешевизна. Действительно, теперь вместо жужжащей и шумящей корзины, превосходящей по размерам и стоимости ПЭВМ, для того чтобы реализовать практически все стандартные функции, необходимые для автоматизации, достаточно вставить в ПЭВМ маленькую плату стандартного размера и использовать прилагаемое к ней стандартное математическое обеспечение. Естественно, что при этом несколько теряется универсальность системы. Однако сейчас выпускаются модульные платы с посадочными позициями, допускающими переконфигурацию системы. Например, на плате, вставляемой в ПЭВМ, имеется набор стандартных посадочных позиций, в которые можно помещать ряд унифицированных модулей (ЦАП, АЦП, мультиплексоры и т. п.). Кроме того, выпускается множество различных плат автоматизации, ориентированных на различные специфические применения. Назовем несколько наиболее распространенных систем такого рода: IBM Data Acquisition, MetraByte, Data Translation, LabMaster. В странах СЭВ и в СССР для ПЭВМ ЕС 1840 начат выпуск комплекта таких устройств, среди которых можно назвать модули EB48, CM607 и др. Технические характеристики рассматриваемых устройств весьма различны. В специализированных платах для высокоскоростного ввода аналоговой информации используются АЦП, имеющие буфер памяти, с частотой преобразования до нескольких мегагерц. В устройствах общего назначения обычно предусмотрены возможности для решения основных задач автоматизации. Рассмотрим типичный состав некоторой средней платы для автоматизации и управления процессами в комплекте ПЭВМ. Она содержит:

- 8 — 256 каналов аналого-цифрового преобразования с разрешением 12 бит и скоростью преобразования 30 — 100 КГц;
- 2 — 8 каналов цифро-аналогового преобразования;
- 2 — 8 24-битных цифровых регистра ввода-вывода;
- 5 независимых 16-битных таймеров.

При этом в большинстве таких систем предусмотрены различные дополнительные возможности: выбор программируемого коэффициента усиления, изменяемый диапазон входных напряжений, счетчики событий и др.

Опишем кратко структуру типичного программного обеспечения такой платы для того, чтобы читатель имел представление о ее возможностях. Для выполнения специальных операций обычно достаточно записать несколько контрольных кодов по определенным адресам памяти или по определенным адресам пространства ввода-вывода. Однако для наиболее частых операций обычно существует пакет программ. Эти программы можно разбить на несколько основных групп.

Процедуры для аналого-цифрового преобразования. Расскажем об основных из них:

ADIN (channel, result)

Эта простая процедура возвращает в программу значение напряжения `result`, прочитанное в канале номер `channel`. Для более сложного считывания последовательности каналов или многих точек используется процедура

ADC (channel, interval, data, length, background)
при вызове которой начинается аналого-цифровое преобразование по каналу `channel` (или по заранее заданному массиву каналов) через интервал времени `interval` в массиве `data`, всего должно быть считано число точек, равное `length`. Процедура считывания может протекать как в фоновом режиме (`background=1`), так и в основном (`background=0`). При работе в фоновом режиме считывание будет происходить без задержки выполнения следующих операторов программы, т. е. можно запустить считывание данных, а в это время рассчитывать какие-либо параметры в программе. При работе в основном режиме выполнение дальнейших операций задержится до тех пор, пока операция считывания не закончится.

Частой задачей также является усреднение данных, считанных по каждому из каналов (если запуск считывания начинается синхронно относительно какого-либо события). Для этого предусмотрены отдельные программы, аналогичные программе **ADC**.

Частой задачей при считывании аналоговых сигналов является запуск считывания после превышения напряжением заданного значения или определение числа пересечений напряжением некоторого заданного уровня. Для этого используются следующие процедуры:

ADCZERO (channel, threshold, data, length, interval)

ZEROCOUNT (channel, threshold, numbercrosses, interval)

Первая из них начинает считывание данных по заданному каналу после превышения входным напряжением порога `threshold`. Вторая производит подсчет числа пересечений за промежуток времени `interval` и выдает результат в переменной `numbercrosses`.

В стандартных пакетах имеется ряд сервисных процедур, например разрешение на запуск считывания с некоторого внешнего устройства, установка коэффициентов усиления по заданным каналам, подсчет времен между двумя последовательно обнаруженными пиками в сигнале или пересечениями заданного уровня. Общее число специализированных процедур для аналого-цифрового преобразования в большинстве пакетов обычно превышает 20, но наиболее необходимы из них обычно две-три программы.

Процедуры цифро-аналогового преобразования. Рассмотрим основные из них. Процедура

DAOUT (channel, value)

преобразует числовое значение `value` в аналоговое напряжение и передает его в канал цифро-аналогового вывода `channel`.

Для вывода массива чисел в различные каналы вывода используется процедура

DAOUTPUT (channel, interval, array, length)

В этой процедуре массив цифровых данных array, имеющий длину length точек, через интервал времени, заданный переменной interval, выдается на выход канала channel. Имеются также модификации этой программы, позволяющие выдавать одновременно целый набор массивов чисел в несколько каналов, заданное число раз повторять в каждом канале вывод того или иного числа и т. д. Например, процедура

DAXYOUT (channelx, channely, arrayx, arrayy, interval, length)

выводит одновременно значения двух массивов, что особенно удобно при работе с двухкоординатными системами, например при управлении перемещением самописца. Для цифро-аналогового преобразования число прикладных программ обычно не превышает десяти.

Процедуры цифрового ввода. Рассмотрим основные из них:

DIBYTE (bank, result)

DIWORD (bank, result)

Эти процедуры читают соответственно байт или слово из регистра bank и помещают результат в переменную result. Для чтения массива чисел применяются другие процедуры, например

DIARRAY (bank, interval, array, length, background)

При вызове этой процедуры происходит чтение массива данных с заданным интервалом времени из заданного регистра. Как и ранее, эта процедура может вызываться в фоновом и в основном режимах.

Одной из основных операций при работе с вводом цифровых данных является анализ переходов каких-либо битов во входном сигнале из 0 в 1 или наоборот. Как правило, такие переходы свидетельствуют об изменении состояния внешнего устройства, подключенного к ПЭВМ. Понятно, что программный анализ подобных переходов является достаточно трудоемкой задачей: необходимо считать цифровой код, выделить из него тот или иной бит и проанализировать его значение. Если такую операцию необходимо делать в реальном времени, то неизбежны потери времени на выделение и анализ бита. Обычно в пакеты программ встраиваются специальные процедуры для такого анализа. Например, программы

BITMASK (bank, bitnumber)

DIO1WAIT (bank, bit)

DIO10WAIT (bank, bit)

позволяют маскировать заданный бит в заданном регистре (при чтении данный бит всегда будет установлен в нуль), находить переходы из 0 в 1 и из 1 в 0 данного бита в данном канале. Обычно имеются также служебные процедуры для подсчета

числа переходов из низкого в высокое состояние битов, для прерывания процедуры чтения и др. Как правило, общее число процедур для цифрового ввода не превышает 10—15.

Процедуры цифрового вывода. Основные из них следующие:

DOBYTE (bank, value)

DOWORD (bank, value)

Эти процедуры обратны описанным выше процедурам для цифрового ввода и посылают байт или слово в выбранный регистр.

Для вывода массива чисел применяется процедура

DOARRAY (bank, interval, array, length, background)

Она аналогична процедуре DIARRAY, описанной выше. Имеются также процедуры, включающие или выключающие один из выходных битов в заданном канале. Например, для некоторых цифровых устройств может быть необходимо, чтобы какой-то бит в процессе работы был всегда равен 1. Тогда вызовом процедуры

DOBITON (bank, bit)

его можно установить в состояние «1», а процедурой

DOBITOFF (bank, bit)

выключить. Важной особенностью цифрового вывода, отличающей его от цифрового ввода, является то, что для управления внешними устройствами может быть необходимо установить какой-то бит или группу битов в определенное состояние на определенное время. Для реализации этой функции предусмотрена процедура

DOSETBIT (bank, bit, time, background)

Она устанавливает заданный бит на время time, причем может работать как в фоновом, так и в основном режимах.

Процедуры таймирования. Рассмотрим основные из них.

Процедура

PAUSE (timeout)

задерживает выполнение программы на заданный промежуток времени timeout.

STARTTIMER (timer)

TIMEREAD (timer, elapsed)

Первая из данных процедур запускает отсчет времени заданного таймера, а вторая считывает прошедшее время в переменную elapsed. Обычно существует также возможность задать интервал времени в определенном счетчике, по истечении которого необходимо запустить ту или иную процедуру. Существуют и более изощренные процедуры, которые позволяют организовать сложное взаимодействие между различными таймерами.

Даже перечисленные здесь простейшие программы дают пользователю богатые возможности по организации управления внешними устройствами и считывания внешних сигналов. Рассмотрим, как с помощью описанных процедур решить две

простые задачи, описанные в самом начале параграфа. Для простоты будем пользоваться Бейсиком.

Задача 1. Координаты точек, которые необходимо просверлить, уже оптимальным образом упорядочены и содержатся в массивах XCOORD(100) и YCOORD(100), аналоговые входы управления координатами сверла соответствуют выходным каналам 1 и 2 по осям X и Y. Сигналом того, что станок закончил перемещение, является появление в цифровом порте 2 числа 32. Предположим также, что при послылке в цифровой порт 1 числа 16 станок начинает процесс сверления, а при послылке в этот порт числа 0 прекращает его. Время на выполнение одной операции сверления составляет 1 с. Тогда всю необходимую последовательность операций выполняет следующая простая программа:

```
FOR X=1 TO 100
  REM Начинаем двигать сверло в текущую точку
  REM Напряжение надо подавать до тех пор, пока движение
  REM не прекратится
  MOVE: CALL DAOUT (1, XCOORD (X) )
        CALL DAOUT (2, YCOORD (X) )
  REM Ждем сигнала об окончании движения
  REM Сигналом окончания является значение 32 во 2 порте
  CALL DIBYTE (2, STOP)
  IF STOP < > 32 THEN GOTO MOVE
  REM Начинаем сверление и ждем 1000 мс до его завершения
  CALL DOBYTE (1, 16)
  CALL PAUSE (1000)
  REM Прекращаем сверление и продолжаем цикл
  CALL DOBYTE (1, 0)
NEXT X
```

Задача 2. Пусть необходимо снять значения напряжения U3 с третьего канала АЦП при изменении напряжения на первом канале U1 от U10 до U11 в зависимости от напряжения во втором канале U2, которое должно изменяться от U20 до U21. Данные поместим в массив ARRAY (U1, U2). Эту задачу решает следующая программа:

```
FOR U2=U20 TO U21
  REM Устанавливаем текущее значение U2
  CALL DAOUT (2, U2)
  REM Начинаем цикл по напряжению U1
  FOR U1=U10 TO U11
    CALL DAOUT (1, U1)
    CALL ADIN (3, ARRAY (U1-U10+1, U2-U20+1))
  NEXT U1
NEXT U2
```

Мы надеемся, что к этому моменту читатель получил общее представление о программировании простых функций, связан-

ных с вводом и выводом информации из ПЭВМ. Поскольку процесс составления программ для автоматизации весьма однообразен и прост, напрашивается естественный вопрос о возможности автоматизации самого этого процесса. Действительно, казалось бы для генерации программы сбора данных и управления устройствами достаточно лишь опросить разработчика о том, какие функции и когда он хотел бы выполнять и в каком виде запоминать данные. На основании полученной информации можно автоматически скомпоновать измерительную программу. Таким образом, можно построить своего рода программируемый протокол обмена с внешними устройствами и избавить пользователя от необходимости составлять программы для автоматизации. В ряде специализированных пакетов, ориентированных на обработку сигналов и съем данных в конкретной ситуации, имеются возможности диалогового задания способа съема аналоговой информации (число каналов, число точек, частота и т. п.). К сожалению, на момент написания книги нам неизвестны достаточно универсальные системы такого рода.

Примером пакетов, включающих в себя функции сбора данных, являются системы ASYST (фирма MacMillan Software), LabTech Notebook (Laboratory Technologies), LABPAC (Scientific Solutions), LTN488 (National Instruments), PC Instruments (Hewlett Packard) и др. В пакетах такого рода обычно имеется система меню, с помощью которой пользователь выбирает способ сбора информации. На рис. 2.1 приведен экран ПЭВМ при работе с таким пакетом. В левой части экрана име-

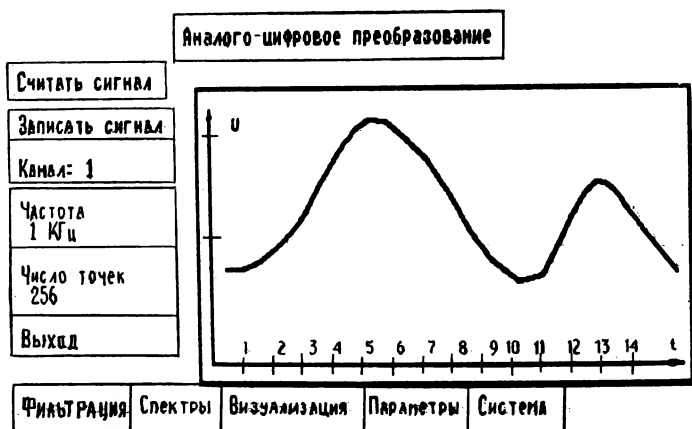


Рис. 2.1. Экран ПЭВМ при работе с системой сбора аналоговой информации

ется меню, по которому пользователь может передвигаться, нажимая на клавиши управления курсором. С помощью этого меню можно выбрать для заданного способа считывания данных (аналого-цифровое преобразование) номер канала, частоту дискретизации, число точек. Справа расположено графическое окно, в котором отображается график измеренных значений.

Применение ПЭВМ в автоматизированных установках несет огромное число скрытых и на первый взгляд неочевидных возможностей. Приведем один простой пример, на наш взгляд хорошо показывающий скрытые возможности компьютеризации задач управления процессами. Предположим, что нам необходимо выбрать оптимальный по некоторому критерию режим работы сложной установки, например системы адаптивной компенсации помех в многоканальной измерительной установке. Мы не можем построить точную математическую модель процесса (он зависит от множества внешних условий, которые нельзя учесть), и процесс характеризуется большим числом внешних переменных, которые можно изменять в тех или иных пределах. Следовательно, наша ситуация такова: имеется черный ящик, на вход которого подается p управляющих сигналов (коэффициенты усиления различных каналов в многоканальной системе), а на выходе снимается некоторый интегральный параметр качества функционирования системы (суммарный уровень шума в регистрируемом сигнале для случая многоканальной компенсации шумов).

По-видимому, любой читатель найдет задачу, близкую его профессиональной области, которая будет напоминать описанную ситуацию. Стоит задача: в изменяющейся, нестационарной обстановке выбрать оптимальные внешние параметры системы. Например, для системы подавления помех критерием оптимальности будет минимальный уровень шумов. Каким образом можно, не имея модели явления и варьируя входные параметры системы, подобрать оптимальный режим? «Ручное» решение подобных задач всегда трудоемко и очень мучительно. Компьютеру справиться с подобной проблемой гораздо проще. Речь идет о решении известной задачи многомерной оптимизации. Даны входные переменные состояния $X_1, X_2, \dots, X_p$, которые в итоге соответствуют некоторым напряжениям или кодам, выдаваемым компьютером на вход установки, и непосредственно измеряемое или рассчитываемое выходное значение функционала Y . Требуется минимизировать это значение. Решим задачу методом прямого покоординатного спуска. Для этого будем перебирать значения каждой входной переменной до тех пор, пока измеряемое Y не станет минимальным, и повторим процесс циклически до тех пор, пока изменения входных переменных на последующих шагах не прекратятся. Задача решена! Сходимость метода можно ускорить и улучшить, применив

один из прямых методов поиска класса сопряженных направлений. Если постоянные времени системы малы, как в случае многоканальной системы подавления шумов, малы, то можно достаточно быстро адаптироваться к изменившейся внешней обстановке и в результате получить адаптивную во времени систему.

В заключение параграфа хотелось бы еще раз остановиться на огромном значении автоматизированных систем на базе ПЭВМ. Из набора простых процедур, обеспечивающих ввод и вывод аналоговой и цифровой информации в ПЭВМ, можно составить целые «симфонии» для управления технологическими процессами, для создания автоматизированных производств и комплексов. Сложность, объем этих программ и одновременное использование математических моделей исследуемых процессов, баз данных и экспертных систем позволяют решить задачи, которые раньше были практически недоступны из-за своей многопараметричности, высокой скорости и больших объемов данных.

2.2. АНАЛИЗ ЭКСПЕРИМЕНТАЛЬНЫХ ДАННЫХ

В предыдущем параграфе мы рассмотрели пути решения задачи сбора экспериментальных данных и ввода их в память ПЭВМ. Однако в 80% случаев сбор данных — это только часть решения проблемы. Основная задача — обработка и анализ сигналов с целью выделения из них интересующей пользователя информации. Методы обработки и анализ экспериментальных данных столь же разнообразны, как и области применения ПЭВМ, и их специфика в значительной степени определяется решаемой задачей. Так, радиоинженеру, исследующему характеристики измерительного тракта, прежде всего необходимо научиться рассчитывать по снятым зависимостям степень искажения сигналов в заданных частотных диапазонах, специалисту по цифровой технике может быть необходимо определять набор параметров принимаемых импульсных сигналов (крутизну фронта и спада), физику-экспериментатору, работающему в ядерной физике, необходимо регистрировать импульсные процессы и рассчитывать их статистические характеристики (например, гистограммы распределения времен между появлениями групп импульсов), врачу-кардиологу необходимо заложить в ПЭВМ специальные программы выделения диагностических признаков из сигналов электрокардиограммы (форму зубцов, наличие «провалов» в сигнале) и т. п. В этом параграфе мы попытаемся дать обзор современных методов обработки сигналов на ПЭВМ и кратко описать их реализацию в существующих пакетах прикладных программ.

Несмотря на огромное разнообразие методов анализа данных, можно выделить несколько основных групп методов:

представление данных,
измерения параметров сигналов,
предварительная обработка и фильтрация,
спектральный анализ и идентификация процессов,
классификация, статистический анализ.

Первые три пункта покрывают до 90% всех приложений методов обработки сигналов на ПЭВМ. Рассмотрим эти методы более подробно.

Представление данных. Первое место по частоте использования и реальной потребности принадлежит группе методов, которые имеют больше отношения к компьютерной графике, чем к обработке сигналов — представлению и визуализации результатов измерений. Надо сказать, что часто математическое обеспечение выпускаемых «специализированных систем» для обработки сигналов или изображений практически на 80% посвящено именно красивой и удобной визуализации данных. Это сделано не зря: если в распоряжении пользователя имеются достаточно мощные и удобные средства для вывода результатов измерений в виде графиков или изображений, если имеется возможность простыми командами изменять масштаб, цвет, ориентацию кривых, то часто отпадает необходимость в дальнейшем более сложном анализе.

Какие основные функции используются для представления графиков? Во-первых, часто возникает необходимость одновременно просмотреть несколько зависимостей. Естественно, что при этом графики должны чем-то отличаться друг от друга. Информацию можно представить различными цветами или использовать несколько окон, в каждом из которых расположен один или несколько графиков. Уже на этом этапе возникает несколько возможностей. Как правило, при анализе семейства кривых пользователь решает две основные задачи: выбор из набора кривых интересующей его для более детального анализа и анализ степени синхронности или взаимосвязи характерных участков кривых в различные моменты времени.

Для решения первой задачи особенно удобно иметь возможность изменить масштаб наблюдаемых сигналов; при этом дисплей ПЭВМ разделяется на несколько окон малого размера, в которых в малом масштабе нарисованы кривые. Проведя «интегральный» анализ графиков, пользователь может увеличить одно из окон до большего размера и более детально исследовать интересующую его кривую. При анализе степени синхронности интересующих пользователя процессов удобно ввести на экране курсор или вертикальную линию, которая может передвигаться по временной оси графика в каждом окне. Для управления таким многооконным графическим интер-

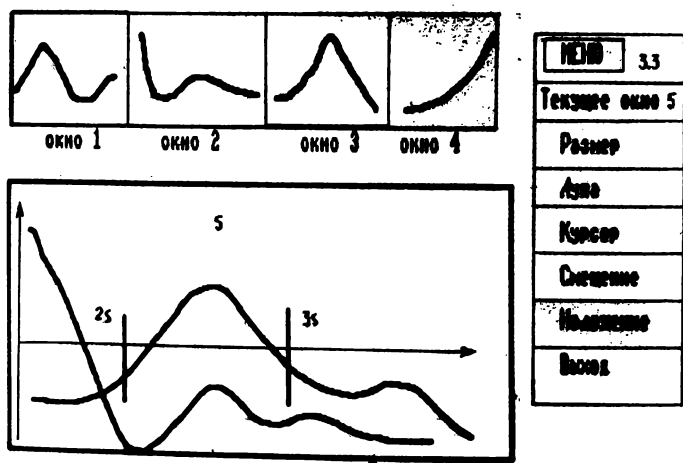
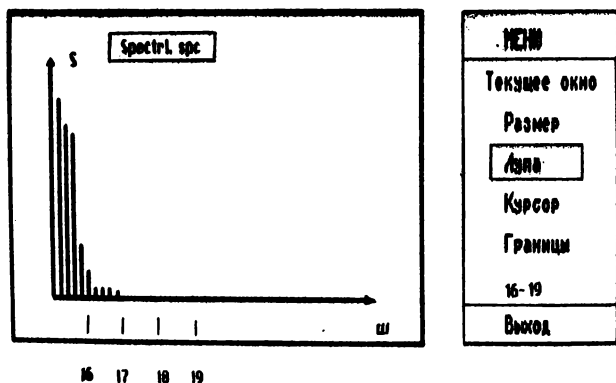


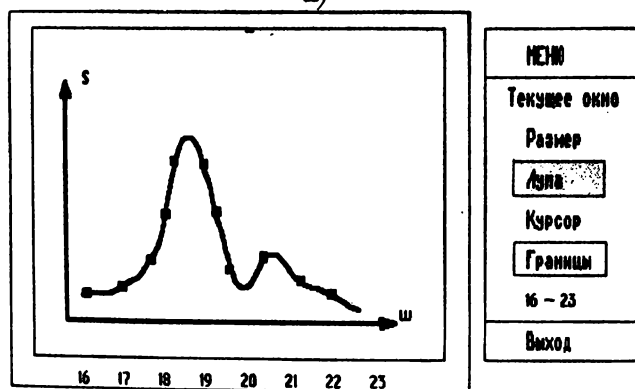
Рис. 2.2. Многооконный графический интерфейс в системе анализа экспериментальных данных

фейсом в стандартных пакетах имеются различные команды: изменения масштаба, цвета, расположения окон, слияние графической информации из нескольких окон в одно и другие. Пример представления набора сигналов в подобной системе приведен на рис. 2.2. В правой части экрана расположено меню, с помощью которого пользователь может управлять визуализацией информации: увеличивать окно, переносить в него кривые. В левой части одновременно видны несколько маленьких окон, в которых записаны различные графики, и большое текущее окно, в котором проводится анализ информации.

Следующий уровень управления процессом визуализации данных реализуется в пределах одного выбранного графического окна, в котором уже оптимальным образом собрана интересующая пользователя информация. Основной функцией здесь, по-видимому, является режим «лупы», позволяющий просматривать часть кривой с изменением амплитудного и временного масштабов. Использование этого режима позволяет выделить из кривой детали, которые могут быть практически незаметны при другом режиме просмотра. Типичный пример такой ситуации — анализ спектров сигналов. Обычно при исследовании достаточно длинной реализации какого-либо процесса в нем возрастает уровень низкочастотных шумов (так называемый шум $1/f$). Мощность низкочастотных составляющих в спектре может в тысячи раз превышать мощность интересующих пользователя высокочастотных составляющих. График спектра сигнала в этом случае выглядит так, как по-



а)



б)

Рис. 2.3. Пример работы с системой анализа экспериментальных данных:

а) график спектра сигнала;

б) выделение фрагмента кривой в режиме электронной лупы

казано на рис. 2.3,а, в котором виден практически только большой пик спектральной плотности вблизи нулевой частоты. На рис. 2.3,б показан растянутый по амплитуде участок высоких частот, на котором ясно видны пики спектра, маскированные на рис. 2.3,а. Другим часто используемым приемом является изменение типа представления данных: введение логарифмического масштаба, использование полярных координат и т. д. Наконец, в пакеты обработки сигналов включаются все функции, которые будут рассмотрены в § 2.4: нанесение на график текста, осей, раскраска, вывод изображения на принтер или графопостроитель и т. п.

Измерения параметров сигнала. В той или иной форме измерения также необходимы практически всем пользователям, связанным с анализом сигналов. Как мы уже упоминали, конкретные измеряемые параметры определяются направлением исследований. В существующем программном обеспе-

чении четко прослеживаются две основных тенденции: создание специализированных программ для измерений конкретных параметров и создание наборов программ для измерения почти всего на свете. У каждого из направлений есть преимущества и недостатки. Первое позволяет быстро и надежно измерять заданные характеристики сигналов, но для расширения возможностей пакета необходимо писать свои собственные программы. Второе направление обычно требует проведения диалога и не обеспечивает высокой точности, но с его помощью можно при необходимости извлечь из сигнала дополнительную информацию.

Рассмотрим простой пример. Предположим, что прикладное математическое обеспечение ориентировано на анализ конкретной ситуации: имеются графики переходного процесса на выходе радиотехнической цепи (рис. 2.4) и необходимо рассчитать следующие параметры: постоянную времени спада, период колебаний и амплитуду первого выброса. Этот пример типичен для анализа различных систем автоматического регулирования. Если известен набор перечисленных параметров, вообще говоря, не представляет особых сложностей написать обрабатывающую программу, которая будет вычислять их значения. Но представим себе, что для анализа качества системы дополнительно потребовалось найти отношение амплитуды второго выброса к первому. Естественно, что в рамках замкнутой системы такое измерение можно сделать только «вручную», например перемещая курсор по кривой и записывая значения на бумагу. Если система допускает возможность программирования, пользователю придется писать дополнительную про-

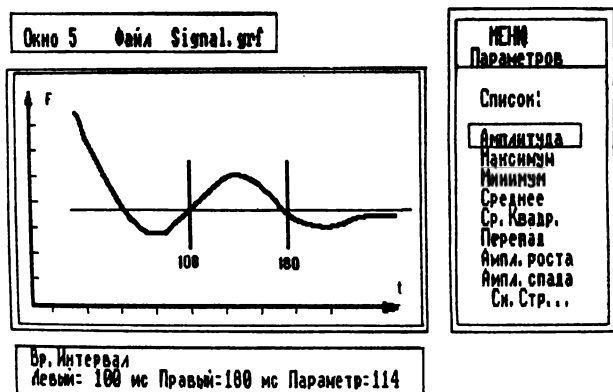


Рис. 2.4. Экран ПЭВМ в системе измерения параметров сигнала

грамму, находящую амплитуду второго выброса и делящую ее на найденную ранее амплитуду первого выброса.

Из только что описанной ситуации существует простой выход: создать достаточно широкий каталог измеряемых параметров и предоставить пользователю возможность выполнения математических операций над измеренными параметрами. На этом принципе построены системы интерактивного измерения параметров сигналов.

Итак, каким образом можно организовать процесс диалогового задания измеряемых параметров? Для этого пользователь должен иметь: 1) широкий набор процедур, каждая из которых измеряет какой-либо конкретный параметр, 2) интерактивную надстройку, с помощью которой указывается, к какому временному отрезку кривой необходимо применить ту или иную процедуру измерения, 3) систему, с помощью которой определяются математические операции над полученными параметрами. Заметим, что в общем случае в подобной системе еще необходима проверка того, что в указанном пользователем интервале времени в анализируемом процессе действительно имеется тот или иной характерный признак. Например, если на участке возрастания кривой пользователь запросит измерение постоянной времени спада, система должна обнаружить невозможность измерения такого параметра и сообщить об этом. Читатель, по-видимому, уже понял, что система для интерактивных измерений параметров сигналов не так проста, как это может показаться с первого взгляда.

Какие же параметры сигнала можно включить в каталог? Эти параметры можно разбить на несколько характерных групп: амплитудные, временные, скоростные и интегральные. Амплитудные параметры — это прежде всего амплитуда характерного сегмента сигнала (импульса, пика), амплитуда изменения сигнала в заданном интервале времени (фактически — перепад за заданное время), максимальное, минимальное, среднее и среднеквадратическое значения сигнала. К временным параметрам относятся постоянные времени возрастания или спада сигнала, времена достижения сигналом стационарного или заданного уровня, времена между характерными точками сигнала (например, период колебаний или время существования импульса амплитуд) и др. Скоростные параметры: максимальная, минимальная, средняя скорости изменения сигнала в заданном интервале времени, максимальный перепад скоростей. Интегральные параметры: площадь под кривой, разность площадей, сумма квадратов значений сигнала. Список этот можно легко продолжить. В прикладных системах число характерных параметров сигнала достигает нескольких десятков.

На рис. 2.4 приведен вид экрана ПЭВМ в режиме измерения параметров системы; с помощью меню пользователь пере-

двигает по графику кривой временные метки и выбирает интересующий его интервал времени. Затем он выбирает из списка параметров интересующий его и дает команду измерения. В специальном окне экрана появляется измеренное значение.

Понятно, что перед имеющим такую систему пользователем открываются широкие возможности по извлечению информации из сигналов. При этом можно, один раз обратившись к процедуре задания необходимых параметров, в дальнейшем многократно использовать подготовленный набор параметров для анализа однотипных данных, а также передавать численные значения получаемых параметров в другие обрабатывающие процедуры, например для классификации процесса.

Обработка сигналов. На этапе предварительной обработки данных начинается собственно обработка сигналов. Под предварительной обработкой понимают различные достаточно простые процедуры, модифицирующие исходный сигнал и приводящие его к виду, более приемлемому для визуального анализа и измерений. Наиболее часто предварительная обработка сигнала необходима для «выкидывания» из кривой ненужной информации, мешающей визуальному анализу. Так, достаточно распространенное явление — медленный низкочастотный дрейф сигнала, на фоне которого происходят информативные изменения амплитуды. При этом результирующий сигнал $f(t)$ можно представить в виде суммы мешающей $d(t)$ и полезной $g(t)$ функций: $f(t) = d(t) + g(t)$.

Для подавления мешающего низкочастотного дрейфа существуют различные процедуры: оценка дрейфа по методу наименьших квадратов, фильтрация данных, приближение данных полиномами и др. В большинстве пакетов для обработки сигналов имеются специальные процедуры для выделения дрейфовой составляющей и вычитания ее из сигнала. На рис. 2.5 приведен пример применения такой процедуры. Другим распространенным явлением является зашумленность сигнала. При этом зарегистрированный сигнал можно представить в виде $f(t) = f(t) + n(t)$, где $n(t)$ — шумовая составляющая. В прикладных системах обработки сигналов обычно реализован широкий набор различных фильтров для сглаживания данных и подавления шумов. Учитывая, что в наиболее развитых пакетах такой набор достигает десятка процедур, мы считаем необходимым дать здесь краткий обзор и сравнение применяемых методов.

Для подавления шума можно применять ряд процедур, простейшей из которых является сглаживание данных. При сглаживании предполагается, что шум имеет более высокочастотную природу, чем сигнал, и, следовательно, подавив часть высоких частот, мы выделим полезный сигнал. На практике обычно применяют сглаживание данных методом скользящего окна. При этом каждая точка сигнала рассчитывается как

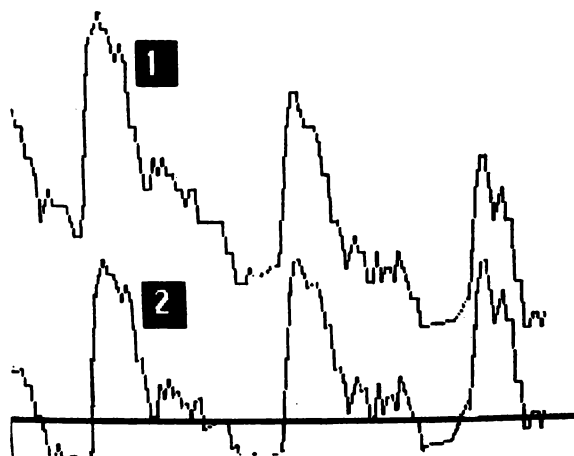


Рис. 2.5. Пример работы системы обработки экспериментальных данных: устранение дрейфа сигнала:
1 — необработанный сигнал, 2 — после устранения дрейфа

среднее по соседним точкам. Например, при сглаживании тройками имеем

$$f(t) = (f(t-1) + f(t) + f(t+1)) / 3.$$

Увеличивая число точек, входящих в среднее, мы все больше «сглаживаем» сигнал, т. е. уменьшаем в нем долю высокочастотных компонентов. Приблизительно тот же результат получится и при многократном применении сглаживания тройками к одному и тому же сигналу. Мы можем управлять степенью сглаживания, задав каждой точке, входящей в усреднение, свой весовой коэффициент. Варьируя этими коэффициентами, например увеличивая вес точек, близко расположенных к центру окна, и уменьшая вес удаленных точек, мы уменьшаем степень сглаживания по сравнению со случаем, когда все точки входят в сумму с одинаковыми весовыми коэффициентами.

Недостатком рассмотренных линейных способов фильтрации является то, что если в исследуемом сигнале существуют резкие перепады амплитуды, то усреднение по соседним точкам сгладит этот перепад и тем самым исказит форму сигнала. В этом случае более оптимальны алгоритмы медианной фильтрации, которые заменяют каждую точку сигнала на среднее в ряду упорядоченных по возрастанию точек из некоторой временной окрестности. Известно, что алгоритмы медианной фильтрации в меньшей степени искажают перепады сигнала. Близкой по эффективности к медианной является адаптивная фильтрация. Основная идея работы адаптивных фильтров крайне проста: например, производится обычное сглаживание сигнала скользящим окном, но ширина окна меняется в зависимости от структуры сигнала: если алгоритм обнаруживает

резкий перепад, то он уменьшает размер окна, на гладких участках сигнала этот размер возрастает.

Если исходный сигнал искажен импульсным шумом, то существуют более эффективные способы борьбы с помехами. Так, учитывая, что импульсы шума должны резко отличаться от амплитуд сигнала, можно создать простой алгоритм, который предсказывает значение сигнала в каждой следующей временной точке, исходя из нескольких предыдущих значений. Если предсказанное и зарегистрированное значения различаются более чем на некоторую заданную величину, значение в точке считается шумовым и заменяется на предсказанное. Популярным методом сглаживания является приближение исходного сигнала полиномом заданной степени или сплайном. В тех случаях, когда можно сделать предположение о спектральном составе шума, оптимальной в смысле минимальной среднеквадратической ошибки является процедура винеровской фильтрации, в которой учитывается информация о характерном поведении энергетических спектров сигнала и шума.

По поводу применения процедур сглаживания сигналов и подавления шумов можно сказать следующее: при небольшом шуме и достаточно гладком сигнале вполне удовлетворительные результаты дает обычное сглаживание методом скользящего окна. Если сигнал при этом имеет достаточно простую и типичную для исследуемой задачи форму (причем целью обработки является измерение каких-либо параметров), то можно рекомендовать приближение исходной функции полиномами. При наличии в сигнале резких перепадов амплитуды желательно использовать медианную или адаптивную фильтрацию. Если известен спектральный состав шума, то лучшие результаты даст винеровская фильтрация. Наконец, если шум имеет импульсную природу, можно использовать процедуру подавления импульсных шумов. Чтобы у читателя создалось впечатление о фильтрации сигналов, на рис. 2.6 показан результат сглаживания зашумленной кривой в системе обработки сигналов.

Довольно часто вместо исходного сигнала пользователя интересует его производная, т. е. скорость изменения амплитуды. Казалось бы, что для вычисления производной достаточно найти разность значений сигнала между последующей и данной точками. Но из-за влияния шумов и погрешностей измерений такая процедура крайне неустойчива: производная, вычисленная таким способом, сильно зашумлена. Существуют различные способы обесшумливания производной, простейшим из которых является предварительное сглаживание исходной функции одним из описанных выше методов. В пакетах обработки сигналов используется несколько различных типов дифференцирующих фильтров, эффективность которых примерно

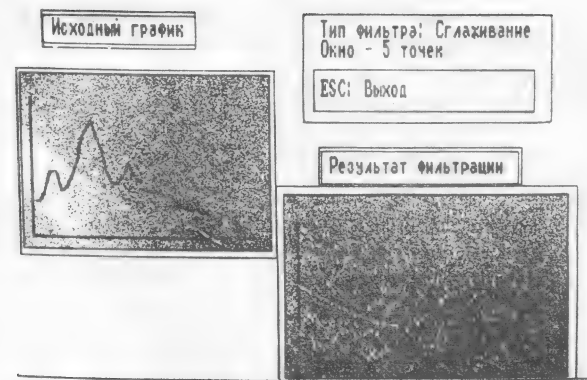


Рис. 2.6. Пример выполнения операции сглаживания сигнала в системе анализа экспериментальных данных

одинакова: дифференциаторы с так называемыми «сигма-факторами», дифференцирующие фильтры Ланцоша и др.

Наконец, к задаче предварительной обработки данных можно отнести задачу фильтрации сигналов. Здесь под фильтрацией мы понимаем выделение из спектра сигнала тех или иных спектральных составляющих. Так, из кривой можно выделить низкие или высокие частоты, вырезать спектральные составляющие в какой-либо полосе, усилить или ослабить выбранные частотные составляющие спектра и т. д. Например, пусть в сигнал попала наводка от сети питания 50 Гц. Тогда мы можем «вырезать» из спектра эту частоту и тем самым очистить сигнал. Другим примером является исследование сложной системы, в которой суммарный вклад в сигнал обеспечивается работой нескольких подсистем с различными характерными частотами (например, при анализе колебаний конструкции, состоящей из нескольких частей с различными размерами и массой). Цифровая фильтрация в этом случае позволяет разделить суммарный сигнал на его составляющие.

Обычно в пакетах прикладных программ имеется несколько процедур фильтрации: низкочастотная, высокочастотная, полосовая и заграждающая. Пользователю предлагается задать параметры, необходимые для выполнения выбранной процедуры. Этими параметрами обычно являются характерные частоты, которые пользователь хочет оставить в сигнале, и тип сглаживающего окна. При низкочастотной фильтрации из сигнала удаляются высокие частоты. Сглаживание шума — это один из примеров низкочастотной фильтрации. При высокочастотной фильтрации подавляются низкие частоты. При полосовой фильтрации пользователь указывает нижнюю и верхнюю частоту в спектре сигнала и в результирующем сигнале оста-

ются только составляющие, относящиеся к оставленной спектральной полосе. Заграждающая фильтрация противоположна полосовой.

Для фильтрации обычно используются процедуры, основанные на вычислении дискретного преобразования Фурье (ДПФ), и методы прямой свертки. Эффективность использования этих процедур зависит от задачи: при малой длине весовой функции (импульсной реакции фильтра) эффективнее прямая свертка, при большой длине — методы, основанные на вычислениях ДПФ. Цифровая фильтрация данных имеет ряд особенностей: из-за дискретного представления данных в ЭВМ возникают неприятности, связанные с появлением паразитных осцилляций данных. Для уменьшения этих осцилляций используют различные типы спектральных окон (Хемминга, Кайзера, фон Ганна, Капеллини и др.). Все перечисленные окна дают примерно одинаковые результаты и пользователь может смело выбирать любое из них.

В стандартных пакетах обработки сигналов обычно реализован также широкий набор рекурсивных фильтров. Рекурсивные фильтры позволяют решать большинство стандартных задач фильтрации. Обычно они более эффективны по объему необходимых вычислений и могут быть применены для обработки сигнала в реальном времени, так как требуют для своей реализации значений сигнала только в текущий и предыдущие моменты времени. Наибольшей известностью пользуются фильтры Баттерворта, которые обычно используются в накладных системах. В пакетах обработки сигналов применяются различные методы для расчета коэффициентов рекурсивных фильтров, углубляться в которые мы не будем.

Спектральный анализ сигналов и идентификация процессов. Во многих задачах обработки сигналов возникает необходимость анализировать спектр исследуемых процессов. В подавляющем числе случаев под анализом спектра понимается его вычисление и визуализация для качественного анализа процесса. Например, при анализе вибраций какой-либо установки пользователя могут интересовать собственные частоты колебаний системы. Эти частоты в спектре сигнала проявляются как пики спектральной плотности. В более сложных приложениях предметом исследования является изменение спектра сигнала при изменениях параметров исследуемой системы. Применение спектрального анализа чрезвычайно разнообразно. Наверное, не найдется области исследований, связанных с получением и анализом любых экспериментальных данных, в которой в том или ином виде не использовалась бы информация о спектре сигнала.

Классическим аппаратом для спектрального анализа является программная реализация преобразования Фурье. В старых пакетах математического обеспечения крупных компьютеров

еще можно найти программы для вычисления спектров, имеющие быстродействие в тысячи раз меньшее, чем сегодня. Такой резкий скачок в этой области связан с появлением БПФ. Для ПК класса IBM PC XT с математическим сопроцессором сегодня время вычисления комплексного спектра сигнала длиной в 1024 точки равно примерно 1 с.

Существует около десятка различных модификаций алгоритмов БПФ: Кули — Тьюки, алгоритм простых множителей, алгоритм Винограда и др. В некоторых пакетах пользователь может выбирать интересующую его процедуру. Различные алгоритмы БПФ незначительно различаются по своему быстродействию (как правило, в 1,5—2 раза) и требуют различной длины сигнала, подвергаемого преобразованию. Наиболее распространен вариант, в котором длина преобразуемого сигнала должна быть равна целой степени двух. При использовании алгоритма простых множителей необходима длина последовательности, равная произведению взаимно простых чисел. Пользователь должен помнить о том, что при вычислении спектра необходима специальная подготовка данных. Например, если длина сигнала равна 80 точками, то ближайшая степень двойки — 128 отсчетов. Для вычисления спектра в простейшем случае недостающие отсчеты дополняются нулями. Этот прием, однако, приводит к появлению в спектре ложных спектральных компонент (в сигнале появляется разрыв). Поэтому используются специальные сглаживающие окна, обеспечивающие плавный переход к нулевым значениям на концах сигнала.

Спектральный анализ используется и для других целей. Так, с его помощью можно выделять скрытые периодичности в сигнале, вычислять корреляционные функции, временной сдвиг сигналов.

Во многие пакеты обработки сигналов включены специальные процедуры для идентификации процессов. Простейшим примером идентификации является измерение передаточной функции какой-либо радиотехнической системы, например канала связи. Для этого на вход объекта подается стандартный испытательный сигнал, на выходе снимается отклик и рассчитываются коэффициенты ослабления различных спектральных составляющих на разных частотах. Так, в канале связи может быть необходимо найти коэффициент ослабления высоких частот с тем, чтобы оценить качество передачи по каналу информации.

Если исследуемая система линейна, то ее полная характеристика может быть получена при подаче на вход короткого импульса (аналога дельта-функции). Для получения частотной характеристики в данном случае достаточно найти спектр отклика системы. Более сложная ситуация существует в ряде систем, в которых имеется несколько входных сигналов. Нако-

нец, при анализе нелинейных систем методы идентификации становятся весьма изощренными: на вход подается шумоподобный сигнал, а выходной отклик системы обрабатывается специальным образом и получают линейную, билинейную и составляющие более высокого порядка передаточной функции. Для анализа ряда нелинейных систем используется степень искажения монохроматического сигнала, подаваемого на ее вход.

В пакеты обработки сигналов включают, как правило, процедуры для оценивания передаточной характеристики линейной системы, ее представления простейшими блоками (линейными звеньями до 2-го порядка), а иногда программы для оценивания нелинейностей при подаче на вход системы колебания определенной частоты.

Классификация, статистический анализ. Классификация является составной частью систем, принимающих решения на основе измеряемых данных. Примером применения классификации может служить распознавание отклонений кардиограмм от нормы, обнаружение неисправностей в различных системах технической диагностики. Математические методы, применяемые для классификации, столь разнообразны, что дать их обзор здесь не представляется возможным. В специализированные пакеты анализа и обработки сигналов обычно включены процедуры классификации, хорошо зарекомендовавшие себя в конкретных областях. В универсальных пакетах обработки сигналов пользователю обычно предоставляется возможность «обучить» компьютер распознаванию на основе выделенных пользователем параметров сигналов. Как может выглядеть такое обучение? Рассмотрим простейший пример.

Вначале пользователь выделяет из сигналов характерные признаки. Эта процедура была описана в начале параграфа. Каждый из признаков имеет свое обозначение, например $A_1, A_2, \dots, A_N$. Затем пользователь входит в специальную страницу меню и указывает, что, по его представлениям, один набор функций принадлежит первому классу, а другой — второму (число классов в общем случае не ограничено). При этом программа классификации вырабатывает правило, по которому она будет классифицировать новые ситуации. В простейшем случае выработка обучающего правила состоит в том, что для каждого из числовых значений признаков подбираются числовые коэффициенты $K_1, K_2, \dots, K_N$ такие, чтобы линейная комбинация $K_1 * A_1 + K_2 * A_2 + \dots + K_N * A_N$ принимала положительные значения для первого класса и отрицательные для второго. Отметим, что в общем случае для надежной классификации число обучающих функций должно быть достаточно велико. Теперь пользователь может запомнить полученное правило классификации, присвоить ему номер или имя и, при получении программой новых данных, вызвать меню классифи-

кации или прикладную программу, которой указывается номер решающего правила. Выходом такой процедуры является решение: данная функция принадлежит такому-то классу. Если решение пользователя по каким-то соображениям не устраивает, он может снова обратиться к обучающей процедуре и т. п.

В ряде случаев возникает необходимость статистического анализа данных: разложение процесса на коррелированные и шумовые составляющие, анализ статистических моментов, предсказание поведения процесса, оценивание плотностей вероятности, проверка статистических гипотез и др. Такие программы также включаются в стандартные системы (см. § 2.3).

По-видимому, читатель уже устал от перечисления различных функций обработки сигналов. Однако надеемся, что он получил представление о богатстве возможностей существующих пакетов. Размеры подобных пакетов (а также их стоимость), обилие возможностей, объем документации производят несколько подавляющее впечатление. Описания некоторых подобных систем занимают до 6 увесистых томов текста, а размер программ — до 50 гибких дисков. Нам приходилось слышать о том, как специалисты, долгое время занимающиеся конкретными вопросами обработки сигналов на ЭВМ, увидев возможности пакетов, были настолько потрясены, что даже хотели прекратить свою профессиональную деятельность. Сложившуюся ситуацию в области программного обеспечения сейчас можно сравнить с периодом перехода от кустарного к мануфактурному типу производства, когда одиночка, как правило, уже не может конкурировать с промышленным производством. Огромное развитие программных средств освобождает пользователя от необходимости самому писать процедуры фильтрации, вычисления спектров и высвобождает его время для решения новых задач на базе имеющегося опыта.

Итак, мы сделали обзор основных методов анализа данных и обработки сигналов, наиболее часто используемых на ПЭВМ. Существует большое число пакетов программ, в которых реализована часть описанных нами функций. Наиболее известны уже упомянутые в § 2.1 пакеты ASSYST фирмы MacMillan Software, пакет LabTech Notebook, пакет BMDP для обработки биомедицинских данных, пакет STATGRAF фирмы Statistical Research и другие. К пакетам обычно прилагаются библиотеки программ, позволяющих выполнять различные процедуры из программ пользователя.

Сделаем одно важное замечание: конечно, хорошо, когда все описанные нами процедуры реализованы в виде библиотек программ, которые можно вызывать из программ пользователя: это фактически рабочий инструмент ученого и инженера. Но создание диалоговых систем такого рода, закрытых для пользователя, в итоге обречено на неудачу: в подавляющем большинстве случаев проще написать программу самому, чем

каждый раз пользоваться глобальной надстройкой. Это совершенно общая ситуация для прикладного программного обеспечения, если оно не предназначено для какой-то совершенно конкретной и узкой цели. Например, с этой точки зрения разумным пакетом обработки сигналов является SPECTRAN, предназначенный для обработки спектров в ядерных исследованиях. В него включен широкий набор узкоспециализированных функций, и поэтому он незаменим в своей области. Существует несколько десятков специализированных пакетов программ: для анализа вибраций, для радиоизмерений, для анализа импульсных сигналов и т. п.

До сих пор речь шла об обработке одномерных сигналов. В последнее время все большее значение приобретают области исследований, связанных с получением и анализом изображений. Обработка изображений играет все большую роль в медицинской диагностике (рентгенография, ультразвуковые изображения, ядерная медицина), в робототехнике (системы машинного зрения, распознающие устройства), в промышленном контроле (анализ качества выпускаемых компонентов и правильности сборки), в неразрушающем контроле. Не будет преувеличением сказать, что не последнюю роль в широком распространении коммерческих систем обработки изображений сыграли именно ПЭВМ. Действительно, еще несколько лет назад приличная система обработки изображений стоила сотни тысяч рублей. Сейчас подобная система, базирующаяся на ПЭВМ, в десятки раз дешевле и тем самым становится доступной для широкого применения. Простой, но красивый пример применения системы обработки изображения на базе ПЭВМ — контроль сборки тормозного устройства на автомобильном заводе. При этом в память ПЭВМ записывается изображение правильно собранного тормозного устройства. На выходе конвейера устанавливается телекамера, сопряженная с ПЭВМ, в память которой вводятся изображения собранных тормозных устройств. Если разность изображений правильно собранной и сходящей с конвейера системы превысит пороговое значение, компьютер выдает сигнал тревоги.

Обработка изображений на ПЭВМ непосредственно связана с наличием специальных дополнительных видеоплат для ввода и визуализации изображений. В настоящее время в мире выпускается несколько десятков таких систем. Наиболее мощные из них имеют разрешение до 4096×4096 элементов изображения и специализированные процессоры для быстрого выполнения графических операций. Например, отечественная видеоплата «Микро СВИТ» для класса IBM PC XT позволяет вводить с телекамеры в ПЭВМ изображения форматом до 1024×1024 элементов отображения с глубиной 8 бит на точку (256 уровней яркости) и отображать это изображение на видеомониторе в 256 уровнях серой шкалы или в любом наборе 256 цве-

тов из палитры в 16 млн. цветов. Для каждой из выпускаемых видеоплат имеется библиотека программ и какая-либо диалоговая система обработки изображений. Рассмотрим основные функции системы обработки изображений на базе ПЭВМ.

Задачи обработки изображений во многом близки рассмотренным нами выше задачам обработки сигналов. Здесь тоже можно выделить фильтрацию с целью подавления шумов или выделения информативных признаков, спектральный анализ, измерение статистических характеристик, программы, связанные с визуализацией и раскраской данных, выделением признаков и классификацией. Рассмотрим типичное математическое обеспечение средней системы обработки изображений на ПЭВМ:

1. Функции ввода, вывода и запоминания изображений. Реализация этих функций в значительной степени зависит от используемой видеоплаты. В типичной системе существует возможность считывания изображения с видеокамеры в телевизионном стандарте и последующего запоминания цифровой картинки на дисках. Существуют системы для сжатия данных при записи изображения на диск. Для поиска изображений в архиве применяются даже базы данных (например, система PICTURE BOOK фирмы CORECO Inc.).

2. Визуализация изображения. При обработке изображений чрезвычайно важен вопрос о выборе способа контрастирования данных и псевдоцветовой раскраски. Например, исследуемое изображение может иметь небольшой динамический диапазон, при этом имеет смысл таким образом преобразовать изображение, чтобы максимально использовать яркостную шкалу. На рис. 2.7 приведено неконтрастированное (а) и контрастированное (б) изображение. Контрастирование осуществляется специальными программами, преобразующими цифровые значения изображения в уровни яркости. Так, на рис. 2.8 показан экран ПЭВМ при работе с программой, которая осуществляет контрастирование. В левой части экрана расположены различные функции преобразования цифрового массива в яркость изображения. Например, линейная функция соответствует обычному преобразованию. Функция F2 отображает только часть динамического диапазона и тем самым производит контрастирование изображения.

3. Измерения на изображении. Простейшие функции измерения на изображениях сводятся к извлечению из кадра профилей яркости, выделения объектов, отличающихся по яркости от фона, вычисления их площадей, средней яркости, минимальных и максимальных значений изображения. В более сложных случаях вычисляется также ряд геометрических параметров объектов: центры тяжести, моменты инерции, периметр, характеристики формы и др.

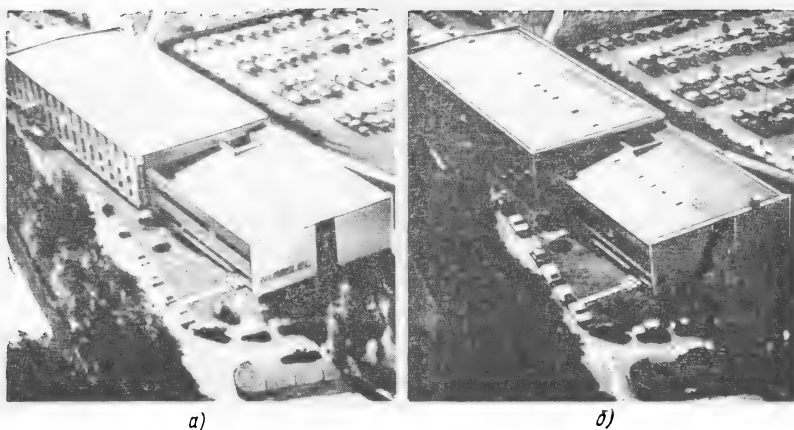


Рис. 2.7. Работа с системой обработки изображений на ПЭВМ:
а) исходное цифровое изображение; б) то же изображение с повышенным контрастом

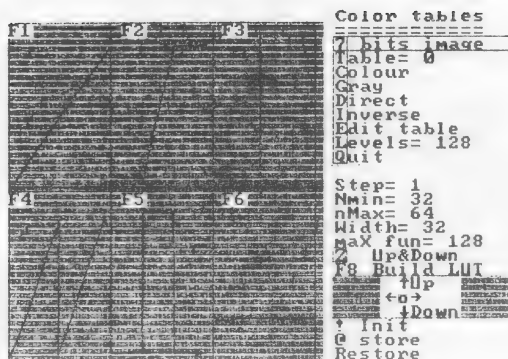


Рис. 2.8. Экран ПЭВМ при работе с системой обработки изображений: работа с меню контрастирования изображений

4. Фильтрация изображения. Для подавления шумов в изображении используется, как правило, двумерное скользящее окно. При этом каждая точка вычисляется как взвешенное среднее по двумерной окрестности точек изображения. В простых пакетах обработки обычно ограничиваются предоставлением пользователю процедуры усреднения по окну размером 3×3 элемента с программируемыми коэффициентами. Но даже используя эту простейшую процедуру, можно путем выбора коэффициентов в окне выполнять более сложные операции: выделение контуров с помощью двумерного дифференцирования, подчеркивание в изображении линий определенного наклона. Например, выбрав коэффициенты в окне 3×3 элемента равными

$$\begin{array}{ccc} -1 & 0 & -1 \\ 0 & 4 & 0 \\ -1 & 0 & -1 \end{array}$$

мы получим двумерный аналог оператора Лапласа, подчеркивающего контуры в изображении. Пример выполнения такой операции показан на рис. 2.9.

5. Статистический анализ изображений. В простейшем случае ограничивается вычислением гистограммы яркости изображения. Гистограмма показывает, сколько элементов изображения имеют тот или иной уровень яркости. В частности, наличие пиков в гистограмме свидетельствует о существовании в изображении областей, имеющих однородную яркость. Искажение формы гистограммы может служить критерием оценивания интегральных характеристик изображения.

6. Классификация изображения. Обычно основана на измеренных характеристиках объектов на изображении. Например, в робототехнической системе может быть необходимо уметь распознавать форму деталей. В простых случаях для этого может быть достаточно вычисления площади и периметра. В большинстве прикладных систем обычно имеются простейшие процедуры классификации, в которых пользователь сам может выделить ряд признаков изображения и объединить их в класс.

Среди пакетов для обработки изображений на ПЭВМ можно выделить системы Microsight, Gray, CompuVision. Существуют также специализированные пакеты программ, предназначенные для анализа в гистологии, обнаружения дефектов поверх-

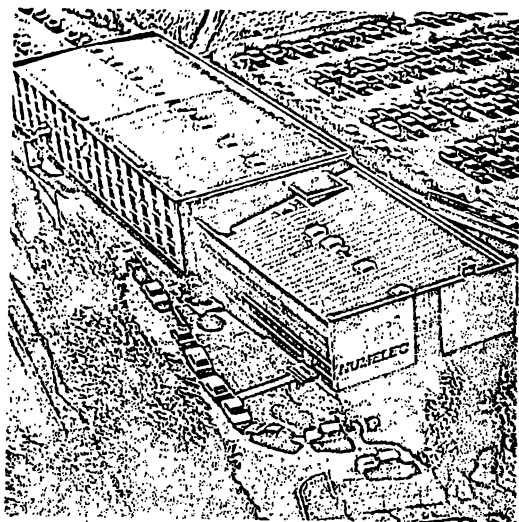


Рис. 2.9. Результат применения оператора Лапласа к цифровому изображению

ностей, контроля качества печатных плат, термографических измерений и др. Число различных пакетов для обработки изображений только на ПК IBM составляет несколько десятков.

2.3. МАТЕМАТИЧЕСКИЕ РАСЧЕТЫ

Практически все пользовательские программы инженерного профиля используют математические расчеты. Инженеру необходимо решать дифференциальные уравнения, вычислять интегралы, находить преобразования Фурье, обращаться матрицы и т. п. Широта приложений здесь огромна: один круг пользователей каждодневно пользуется сложными статистическими методами, другой решает дифференциальные уравнения, третий связан с задачами многомерной оптимизации и т. п. Для того, чтобы удовлетворить эти разнообразные запросы, для ПЭВМ разработано различное математическое обеспечение. Это обеспечение можно приблизительно разделить на три основные категории:

библиотеки программ для математических расчетов — набор подпрограмм численного анализа;

специализированные пакеты для решения конкретных математических задач: специализированные системы для статистического анализа, решения дифференциальных уравнений и др.;

декларативные языки и системы для математических вычислений: специализированные языки программирования и диалоговые системы, позволяющие формулировать и решать расчетные задачи максимально естественным образом.

Рассмотрим это математическое обеспечение более подробно.

Библиотеки программ для математических расчетов. При решении задач инженером или ученым далеко не всегда удастся найти решение в рамках стандартных прикладных систем. Более того, если прикладные системы общего назначения и позволяют решить задачу, то нередко эффективность работы с ними невысока, например из-за перегруженности диалогом. Таким образом, четко выраженная тенденция использования ПЭВМ состоит в том, что в большинстве случаев (иногда после того, как отработана методика решения конкретной задачи с помощью подходящей прикладной системы) пользователь начинает составлять свои собственные программы. Хорошо, если необходимые процедуры доступны в виде библиотеки подпрограмм, прилагаемой к прикладной системе (как, например, в пакетах обработки сигналов). Если это не так, то инженер вынужден начинать разработку известных процедур самостоятельно.

Наибольший контингент пользователей ПЭВМ, разрабатывающих программы, безусловно заинтересованы в использова-

нии библиотек программ для численного анализа. Мы настоятельно рекомендуем читателям не терять впустую время и попытаться разыскать для своей ПЭВМ одну из таких библиотек. Конечно, некоторые процедуры численного анализа достаточно просты и могут быть легко реализованы в виде программы на каком-либо из языков высокого уровня. Например, ничего не стоит написать программы для вычисления гистограммы распределения или для перемножения двух матриц. Написать процедуру обращения матрицы значительно сложнее, не говоря уже о многомерной оптимизации, решении дифференциальных уравнений и т. п. Дело не только в том, что такие программы объемны и пользователь может потратить длительное время на изучение возможных алгоритмов, написание и отладку программ. Численный анализ является довольно сложной областью, при реализации многих алгоритмов существуют, на первый взгляд, неочевидные трудности, а эффективность некоторых процедур, реализованных «в лоб» на ЭВМ, может быть в сотни раз ниже, чем при их оптимальном решении. Хороший пример из этой области — реализация преобразования Фурье. Авторы однажды столкнулись с ситуацией, когда один из пользователей, которому нужно было рассчитать Фурье-преобразование функции, написал программу, которая честно подсчитывала сумму ряда Фурье. При прямом вычислении интеграла Фурье на ЭВМ объем вычислений пропорционален $N \times N$, где N — длина преобразовываемой функции, а при вычислении с помощью быстрых алгоритмов этот объем уменьшается до $N \times \log N$. Пусть N равно 1000, тогда при прямых вычислениях придется выполнить более миллиона арифметических операций, а во втором случае — всего десять тысяч! Естественно, что этот человек был потрясен, когда увидел, что при обращении к стандартной библиотеке его программа начала рассчитывать результат за 2 с вместо 3 мин.

Доступны различные библиотеки численного анализа для ПЭВМ. Они делятся на специализированные библиотеки, разработанные для решения задач в какой-то специальной области (например для решения интегральных уравнений), и на библиотеки общего назначения. Из последних наиболее известны пакеты SSP (Scientific Subroutines Package) и NAG (Numerical Algorithms Group). Пакет SSP был разработан для больших ЭВМ и уже стал классическим. Он просто перенесен на ПЭВМ с больших машин, например с машин серии ЕС, СМ-4, и др. Пакеты типа SSP и NAG состоят из библиотек подпрограмм. Эти библиотеки разделяются на несколько групп:

статистический анализ: факторный анализ, дискриминантный анализ, анализ временных рядов, генерация случайных чисел и т. д.;

матричные операции: транспонирование, обращение, вычисление собственных значений и векторов, сложение, вычитание, решение систем линейных уравнений;

вычисление специальных функций: Бесселя, гамма-функции, полиномы Лежандра, Лаггера, интегралы Френеля и т. д.;
вычисление интегралов Фурье;

интегрирование функций (квадратурные формулы, метод Гаусса, трапеций и т. д.);

решение дифференциальных уравнений различными методами (Рунге — Кутта, предиктор-корректор и др.);

решение систем нелинейных уравнений;

многомерная оптимизация (методы Ньютона, сопряженных градиентов и др.);

полиномиальные операции;

сортировка;

Рассмотрим простой пример работы с этим пакетом. Пусть в библиотеке подпрограмм имеется процедура обращения матрицы методом Гаусса — Жордана. Тогда пользователь просто включает в свою программу на Фортране вызов

CALL MINY(A, N)

где A — исходная матрица размеров $N \times N$, которая после вызова программы заменяется на обратную. Теперь при выполнении операции сборки программы пользователь должен указать соответствующую библиотеку, например MATRICE.LIB.

Кроме пакетов SSP и NAG, существует большое число более узко ориентированных пакетов для численного анализа. Среди них можно назвать пакет MicroWay, в котором реализован набор операций для работы с большими матрицами, системами линейных уравнений большой размерности и реализован ряд эффективных процедур для выполнения преобразования Фурье. Этот пакет является одним из наиболее эффективных с точки зрения быстродействия, так как он написан с учетом особенностей программирования математического сопроцессора 8087.

Специализированные пакеты для решения различных математических задач. Широкое распространение получили различные пакеты программ для моделирования решений дифференциальных уравнений. Наиболее известен среди них пакет Phaser, вышедший отдельной книгой с приложенными дискетами в издательстве Springer. С помощью этого пакета можно моделировать решения различных дифференциальных уравнений. Он состоит из следующих основных меню:

задания типа дифференциального уравнения, начальных условий, выбора численных параметров и способа численного решения;

выбора способа представления результатов;

графического манипулирования полученными данными.

В меню задания типа дифференциального уравнения пользователь может записать с помощью простых обозначений систему дифференциальных уравнений или выбрать один из примерно ста типов стандартных дифференциальных уравнений, имеющихся в библиотеке пакета. При этом существует возможность выбрать размерность уравнения, задать число параметров и констант, начальные условия, шаг интегрирования, способ численного решения (метод Эйлера, Рунге — Кутта, разностные схемы). В библиотеке стандартных дифференциальных уравнений пакета имеются практически все наиболее известные системы дифференциальных уравнений: линейные и нелинейные (например, такие классические уравнения, как осциллятор Ван Дер Поля, уравнение Хопфа, набор уравнений теории катастроф (сборка, складка, гиперболическая омбилика), уравнения Эйлера и Бесселя, Лежандра, Матье и др.).

В меню выбора способа представления результата пользователь может выбрать представление данных в виде фазового портрета, потока, поля направлений, отображения Пуанкаре, последовательности итерационного приближения к решению или совместить несколько способов представлений в различных окнах экрана.

В меню графического манипулирования данными пользователь может менять масштаб полученного изображения по осям, вращать трехмерную картину, представлять данные в перспективе и в проекциях.

Пакет Phaser позволяет быстро и эффективно представить вид решений дифференциального уравнения, проследить влияние параметров на решение, оценить степень устойчивости решения, найти критические точки и области неустойчивости колебательных систем и многое другое. Работа с этим пакетом увлекательна не только для инженера, проверяющего, как себя будет вести то или иное дифференциальное уравнение, но и для математика, исследующего поведение сложных нелинейных систем. На рис. 2.10 показан экран ПЭВМ при работе с пакетом такого рода: рассчитан фазовый портрет уравнения Даффинга.

Большое распространение получили также пакеты для статистического анализа данных. Из наиболее известных можно назвать следующие: Exec*U*Stat, The Forecasting Edge, MicroStat, 4CaSt/2, Soritec, SPSS/PC, STATA, уже встречавшийся нам Statgraphics, Statpack Gold, Statpro, StatView, Systat, WormStat и др. Эти пакеты имеют различную структуру. Например, пакет Statgraf содержит практически все мыслимые функции статистического анализа. Другие пакеты, например MicroStat, более просты в обращении и включают лишь ряд функций для оценивания плотностей распределений, выполнения регрессионного анализа и т. д.

Но любой из перечисленных пакетов намного перекрывает реальные возможности рядового пользователя по программ-

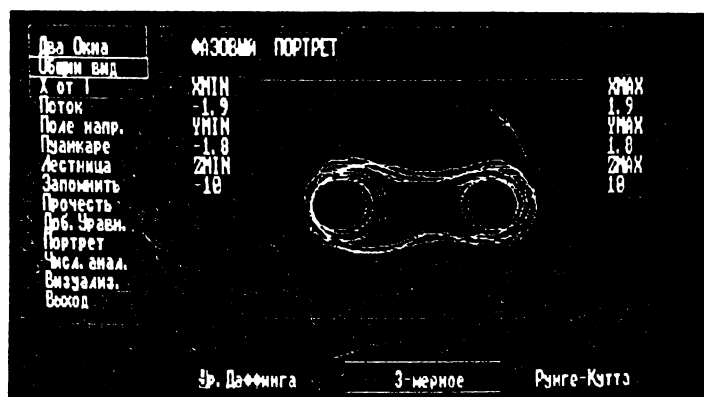


Рис. 2.10. Экран ПЭВМ при работе с системой моделирования решений дифференциальных уравнений

рованию распространенных функций статистического анализа. Подробно рассказывать про работу этих пакетов просто не имеет смысла: каждый из них снабжен набором меню, встроенными подсказками, высококачественной графикой. Более того, некоторые из перечисленных пакетов, например SPSS/PC, имеют встроенный язык программирования, с помощью которого пользователь может написать специализированную программу для выполнения последовательности расчетных операций.

Широкое внедрение ПЭВМ в математические расчеты и возрастающее число пакетов прикладных программ привели к появлению новых терминов типа «Экспериментальное моделирование динамических систем», «Рабочая станция для экспериментального моделирования», «Экспериментальное статистическое моделирование». Учитывая, что этот процесс пока только начинает развиваться, следует ожидать изменения подхода к исследованию широкого класса математических задач, а также к обучению.

Декларативные языки для математических вычислений. Предназначены для максимального облегчения процесса вычислений и построения математических моделей. Действительно, программирование расчетной задачи сводится к тому, что инженер переводит формальные математические выражения на язык ЭВМ, при этом учитывая набор формальных правил и ограничений конкретного языка программирования. Эту задачу было бы разумно поручить компьютеру, выработав удобный способ ввода информации о задаче, напоминающий запись математических формул на листе бумаги. В идеале системы такого рода должны свестись к тому, что человек записывает на экране математическую задачу (например формулы, уравнения, неравенства, формальные правила) и получает ее решение в графи-

ческой или цифровой форме. До такого уровня, конечно, еще далеко, но многие из уже разработанных систем значительно упрощают вычисления на ПЭВМ.

Наиболее распространены системы для интерактивного проведения вычислений. В большинстве таких систем такого рода пользователю предоставляется возможность задать:

поле переменных задачи (определить условные обозначения элементов, задать их начальные значения, единицы измерения и ограничения на допустимые значения переменных);

поле правил, в котором записываются математические операции между заданными переменными (арифметические и логические выражения, условные переходы);

способ решения задачи (прямые расчеты, итеративный поиск решения, оптимизация по параметру методом многомерного поиска);

способ представления результатов (графический, числовой, таблицы и т. п.).

При работе с системой пользователь выбираем из главного меню поле, в которое он хотел бы ввести информацию. Каждое поле системы — это таблица, обычно занимающая часть экрана ПЭВМ. Различные таблицы могут быть видны одновременно. При этом пользователь задает переменные, записывает математические операции, которые необходимо произвести между этими переменными, тип решаемой задачи, способ выбранной информации. Теперь он может варьировать начальные значения переменных и одновременно видеть результаты вычислений. Рассмотрим конкретный пример: необходимо рассчитать частотную характеристику колебательного контура с различными индуктивностью, емкостью и сопротивлением. Таблица переменных в этом случае выглядит следующим образом:

Таблица переменных

Начальное значение	Обозначение переменной	Результат	Комментарий
	L		Индуктивность
	C		Емкость
	R		Сопротивление
	W		Резонансная частота
	A		Затухание
	B		Полоса пропускания
	F		Частота

Таблица правил записывается следующим образом:

Таблица правил

Но- мер	Правило	Комментарий
1	$W = (1/LC)^{0.5}$	Резонансная частота
2	$K = 1 / (A(1 + 4/A^2((F-W)^2)))^{0.5}$	Частотная характеристика
3	$B = WA$	Полоса пропускания
4	$A = R(C/L)^{0.5}$	Затухание

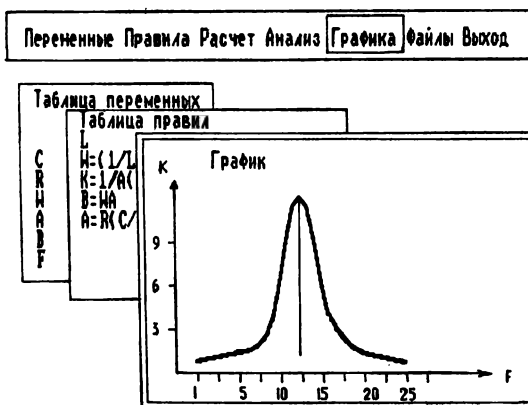


Рис. 2.11. Работа с системой интерактивных вычислений на ПЭВМ: результат расчета частотной характеристики колебательного контура

Теперь пользователю достаточно задать в таблице переменных числовые значения L , R , C , определить значения частоты или указать диапазон их изменений, и он получит в графическом окне частотную характеристику (рис. 2.11). В таком режиме мы просто используем ПЭВМ как интеллектуальный калькулятор, делающий вычисления по формулам. Однако в рамках даже такого простейшего примера перед пользователем открываются богатые возможности. Например, сотрем значения L , C и зафиксируем полосу пропускания. При этом в таблице переменных появятся подсчитанные значения L , C и резонансная частота. Естественно, что при этом значения L , C будут взяты «с потолка» — система найдет первое возможное решение. Теперь пользователь может задать в специальной строке таблицы ограничения на возможные значения L и C и система начнет поиск решения, удовлетворяющего этим ограничениям. Пользователь может ввести в систему дополнительные логические условия, например: «Если сопротивление больше, чем R_0 , то резонансная частота не должна превышать W_0 ». Для этого в таблице правил мы дописываем следующую строчку:

5 IF ($R > R_0$) THEN $W \leq W_0$

Этот процесс можно неограниченно наращивать. Гибкость и удобство задания различных ограничений и модификаций модели процесса впечатляют. Читателю, видимо, понятно, что если он напишет программу расчета частотной характеристики контура, то для каждого из описанных здесь расширений ему необходимо будет модифицировать процедуру расчета. Возмож-

ности системы, однако, не ограничиваются перечисленными: с ее помощью можно в диалоговом режиме проводить оптимизацию параметров. Например, в рамках нашего примера пользователь может дать системе следующее задание: «Максимизировать полосу пропускания с учетом введенных ограничений на затухание и резонансную частоту». Это может быть сделано, если в специальном окне «Тип расчета» мы введем следующую запись:

MAX(B)

при этом сохранив таблицу правил.

В таблицу правил системы можно включать любые математические выражения. В наиболее сложных системах можно использовать различные процедуры численного анализа, в том числе:

- задавать связь между переменными в виде дифференциальных уравнений;

- описывать переменные как результат вычисления интеграла;

- включать вызовы любой процедуры из пакета подпрограмм численного анализа или процедуры, созданной пользователем системы.

Например, в рамках заданной модели контура можно ввести в таблицу правил дифференциальное уравнение, описывающее переходный процесс в электрической цепи и рассчитать вид этого переходного процесса. Фактически возможности системы такого рода очень велики: если человек может задавать математическую модель, то он получит все результаты, ни разу не прибегнув к программированию. Ему необходимо только запомнить относительно простой набор правил записи математических выражений и «набить руку» в работе с такой системой.

Обычно в системы математических вычислений введены развитые средства диагностики. Например, если в рамках введенных ограничений не удастся найти решения задачи, то пользователю будет сообщено об этом.

Практика показывает, что с помощью подобной системы удастся решить более половины типичных расчетных задач, встающих перед инженером, при этом процесс решения задачи ускоряется в 5—10 раз. Одним из таких наиболее распространенных пакетов для ПЭВМ является программа TK!SOLVER фирмы Software Arts.

Серьезной задачей при создании систем для математических вычислений является выработка языка, на котором математические выражения записывались бы в максимально естественной форме. Эта задача решается различными способами: от создания библиотеки функций, запись которых напоминает обычные правила записи математических выражений, до построения специальных систем символьных вычислений. Среди них наиболее известны системы Mu-Math и Reduce. С их помощью

можно производить математические преобразования в символьном виде. Распространены также специальные пакеты программ для ПЭВМ, в основном для решения различных уравнений, такие, как Calfix (Dynacom Inc.), Eureca, MathCAD (MathSoft Corp.), MATLAB и др. Пользуясь этими системами, можно выполнять самые различные математические преобразования. Например, в рамках системы mu-Math пользователь может вести с ПЭВМ примерно следующий диалог (стрелка соответствует операции возведения в степень):

EXPAND $(w+1)^2 - w^2/4$; (раскрыть скобки и записать результат)

@: $1+2w+3/4w^2$ (ответ ЭВМ)

Другая инструкция языка: преобразовать частное к произведению двух сомножителей. Для этого достаточно записать

DIVOUT $(w^2+2w+1)/(w^2-1)$

и получить ответ

@: $(1+w)(1-w)$

Читатель, наверное, уже понял, что иметь под рукой такой интеллектуальный калькулятор, производящий выкладки, чрезвычайно удобно в тех случаях, когда по роду работы приходится иметь дело с различными расчетами. Компьютер не сделает глупых ошибок и произведет преобразования быстрее человека. Наконец, такую систему удобно использовать и для проверки правильности формул, полученных традиционным путем, с помощью карандаша и бумаги.

Система может работать с тригонометрическими функциями, логарифмами, умеет производить разложение в ряды, работать с полиномами, дифференцировать и интегрировать, в общем делать огромное число математических операций в символьном виде. Скорость их выполнения не всегда высока, но ПЭВМ, по-видимому, все равно произведет выкладки в несколько раз быстрее хорошо успевающего студента. Кроме того, компьютер не сделает ошибки при сокращении различных членов.

Приведем еще несколько примеров работы с системой. Например, мы хотим посчитать вторую производную по переменной x от введенного выражения. Для этого пишем системе

DIF $(ax^3+c\cos(xy), 2, x)$

и получаем ответ

@: $6ax - cy^2 \cos(xy)$

Различные системы символьных преобразований позволяют делать и более сложные операции — решать линейные системы уравнений и дифференциальные уравнения. Например, пользователь может записать на экране дифференциальное уравнение и потребовать его решить. В том случае, если решение может быть получено в аналитической форме, оно будет выдано на экран с сохранением принятых правил записи. В противном случае решение получается с помощью процедур численного

анализа и представляется в графической форме. Здесь пользователь может изменить значения коэффициентов, потребовать построить новое решение и т. д.

Освоить систему типа *mu-Math* довольно просто еще и потому, что вместе с ней поставляются обучающие дискеты с интерактивными уроками.

Для того чтобы научиться работать с такой системой, надо затратить несколько дней: в системах символьных вычислений имеются свои ограничения и сложности, ряд соглашений, словом, для того чтобы эффективно ее использовать, надо «набить руку». Но если пользователю приходится часто выполнять механические вычисления типа взятия производных или разложений в ряды, то эта работа себя оправдывает.

2.4. МАШИННАЯ ГРАФИКА

В этом параграфе мы попытаемся кратко описать возможности графики ПЭВМ, реализованные в доступных стандартных пакетах общего назначения и в языках программирования. Заметим, что специфические приложения, такие, как системы графического проектирования и средства графики, включаемые в интегрированные системы, будут рассмотрены отдельно. Но даже графических систем общего назначения для ПЭВМ в настоящее время такое множество, что простое перечисление прикладных пакетов программ заняло бы все место, которое мы можем отвести для этого параграфа. Вышел целый ряд книг и руководств по машинной графике, существует ряд международных стандартов и т. п. Да и задачи, требующие графического воплощения инженером, чрезвычайно разнообразны и очень зависят от конкретной области работы. Возникает вопрос: можно ли выделить в этой массе приложений, пакетов программ какие-то общие моменты, которые окажутся полезны практически каждому пользователю ПЭВМ?

Для того чтобы ответить на этот вопрос, достаточно просмотреть любую статью, учебник, монографию в любой из технических областей. Если отбросить чертежи и схемы, которые мы только что отнесли к специфическим областям графики, то не менее 80% остальной графической информации представлено в форме кривых или семейств кривых, диаграмм, а 15% представлено в форме поверхностей или изображений. Оставшиеся 5% приходятся на диаграммы типа разброса одного параметра относительно другого. Весь опыт работы авторов показывает, что в первом приближении любой пользователь ПЭВМ, применяющий ее для расчетов, моделирования, автоматизации, полностью удовлетворен, если он может представить получаемую информацию в форме графика (в крайнем случае нескольких графиков одновременно). Если

пользователь вдобавок может изменить масштаб и цвет графика, удобно скомпоновать несколько кривых на экране и пересчитать шкалы, то, как правило, можно быть уверенным, что более изощренные средства машинной графики понадобятся ему еще не скоро.

Для ПК разработано большое число интерактивных графических пакетов. Среди них можно встретить и довольно разумно организованные, умеющие выполнять небольшое число часто встречающихся функций, и глобальные, которые позволяют сделать любую мыслимую операцию и при этом, как всегда, перегружены диалогом и занимают почти всю оперативную память. К сожалению, большинство этих пакетов «закрыты» для пользователя. Это означает, что они не содержат библиотеки программ, которые можно было бы вызвать из программы пользователя, а используют данные, подготовленные внешней программой и записанные в файл в определенном формате, причем этот формат записи для различных пакетов сильно различается. Другая крайность — пакет прикладных программ с интерфейсом для различных языков программирования без интерактивной надстройки.

Ярко выраженная тенденция в использовании графических систем на ПЭВМ заключается в том, что пользователь предпочитает включать вызовы графических процедур в свою программу и редко применяет специализированные графические системы. Это легко понять, если учесть, что обычно графика необходима в процессе работы прикладных программ для немедленного визуального контроля результатов. Однако специализированные пакеты предоставляют множество сервисных возможностей, которые не так просто реализовать в своей программе: гибкое масштабирование, выбор цветов, удобную разметку осей, включение в поле графика текстовой информации и т. д. Поэтому реальная тенденция при использовании графики такова: большинство графических функций встраивается непосредственно в прикладные программы, а специализированные графические пакеты применяются для удобства представления результатов, обычно при подготовке отчетов, статей и т. п.

Попытаемся вначале описать работу с интерактивными графическими пакетами программ. Несмотря на то, что таких пакетов множество, работа с ними в значительной степени однотипна. Из наиболее распространенных можно назвать пакеты Energraph фирмы Enertronics Research, Fast Graphs фирмы Innovative Software, Scientific Plotter фирмы Interactive Microware. При вызове этих пакетов на экране появляется главное меню, показывающее функции, которые можно выполнить. Среди этих функций можно выделить следующие: загрузка, модификация и запоминание данных,

выбор типа графики и ее построения,
выбор параметров графика,
наложение на график текста,
редактирование графического изображения,
служебные функции.

Рассмотрим их более подробно.

Загрузка, модификация и запоминание данных. Служат для подготовки числовой информации, которая будет изображаться в виде графика. Эти данные могут быть заранее записаны в файл какой-то другой прикладной программой или введены с клавиатуры в ходе предыдущей работы с графическим пакетом. Обычно при вызове этого раздела меню пользователю предлагается несколько возможностей: просмотреть каталог файлов на диске, выбрать необходимый файл, просмотреть числовые значения, изменить формат данных (например, перейти от плавающей точки к целым числам и т. п.). Пользователь может изменить какие-либо числовые значения и записать модифицированные данные на диск. Одной из возможностей является ручной ввод значений в специальную электронную таблицу или какое-либо математическое преобразование исходного массива данных (например, логарифмирование).

Выбор типа графики и ее построения. Когда данные для графика готовы, пользователь может перейти к выбору типа графического представления. Даже чисто одномерные данные можно представить различными способами: соединяя последовательные точки отрезками прямой линии или сегментами параболы, выделяя узловые точки графика специальными символами, вообще не соединяя точки графика, строя в каждой точке столбики, имеющие различную ширину, высота которых соответствует данному числу, наконец, строя круговую диаграмму, угловой размер секторов которой соответствует заданным числовым значениям. Можно для удобства комбинировать различные виды представления. Наконец, программе необходимо указать, нужно ли построить зависимость одной функции от другой, необходимо ли построить некоторый двумерный график, нужно ли нарисовать одновременно несколько кривых и т. п.

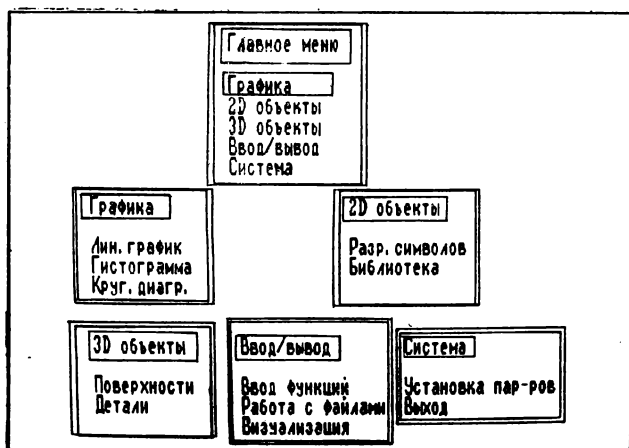
Выбор параметров графика. Функции выбора параметров графика также весьма разнообразны. Пользователь должен указать типы шкал графика, тип разметки (ставить ли метки по осям или нет, провести ли семейство ортогональных линий в местах расположения меток, какие числовые и текстовые значения разместить вдоль координатных осей), цвета графика и фона, размеры и положение графика на экране. Читатель может легко понять, что диалог с системой может растянуться до бесконечности. Во всяком случае, для построения одного графика в большинстве интерактивных систем приходится ответить на 20—30 вопросов.

Наложение на график текста. Предположим, что весь этот цикл пройден. Теперь нам предоставляется возможность наложить на готовое изображение произвольный текст. Например, мы построили график зависимости напряжения на выходе схемы от времени. Тогда можно в красивой рамке написать «Зависимость напряжения на конденсаторе C15 от времени при подаче на вход прямоугольного импульса». Вдоль оси ординат можно написать «Ось времени», вдоль оси абсцисс «Напряжение в вольтах».

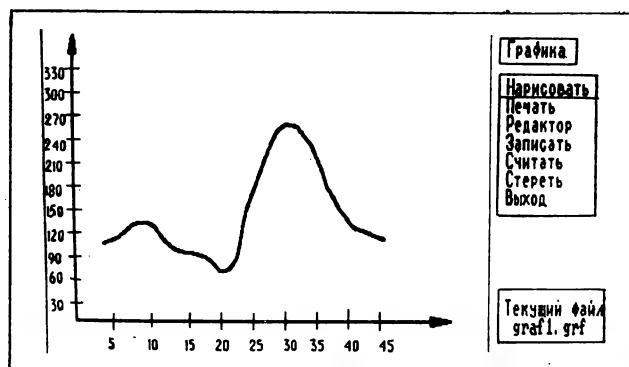
Редактирование графического изображения. При желании можно провести графическое редактирование полученного изображения. Вообще говоря, графические редакторы — это отдельный класс программного обеспечения для ПЭВМ и они частично будут рассмотрены в § 2.5, посвященном системам автоматизированного проектирования на ПЭВМ. Здесь отметим только, что теперь мы можем внести любые изменения в полученное изображение: при выходе в режим редактирования изображения на экране возникает курсор и, передвигая его, можно рисовать отрезки линий, круги, прямоугольники, отрезки кривых, закрашивать полученные фигуры цветом или заполнять их некоторым фоном (штриховкой определенного цвета, текстурой и т. п.). Одним из основных применений графического редактора при подготовке графиков является выделение каких-либо событий в кривой, например выделение отрезками линий заданного интервала времени.

Служебные функции. К служебным функциям относятся: выбор диска, на котором хранятся данные, инициализация типа экрана и печатающего устройства, распечатка графика на принтере, построение на графопостроителе и др.

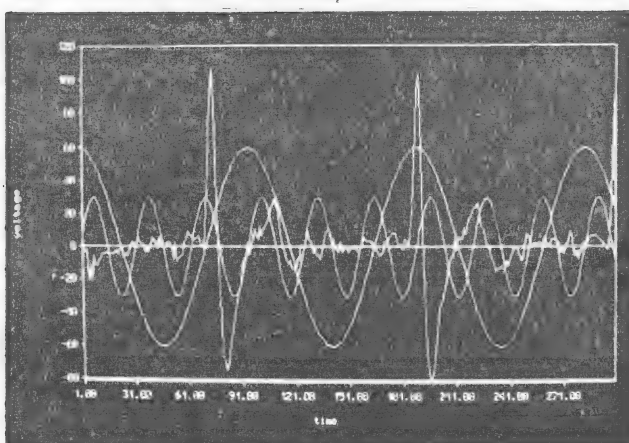
К сожалению, задание параметров графики всегда будет оставаться длительной и утомляющей процедурой. Единственным разумным выходом из этой ситуации можно считать создание набора специальных описателей типов графика. Пусть в конкретных исследованиях пользователь имеет дело с несколькими видами графиков. Тогда системой, близкой к идеальной, являлась бы такая, в которой: 1) один раз вызывается интерактивный графический пакет, в котором пользователь создает специальное описание графических параметров (разметка, расположение графических окон, цвета и т. д.); 2) при последующих вызовах пакета специальным образом указывается, в соответствии с каким именно описанием будет строиться тот или иной график, и соответствующие шаги диалога опускаются; 3) описание параметров графика, построенное в ходе использования пакета, можно включать в вызовы стандартных графических программ из программы пользователя. В некоторых графических пакетах пересмотрены функции стандартного задания параметров графика. При этом весь рассмотренный выше диалог для однотипных данных можно провести один раз,



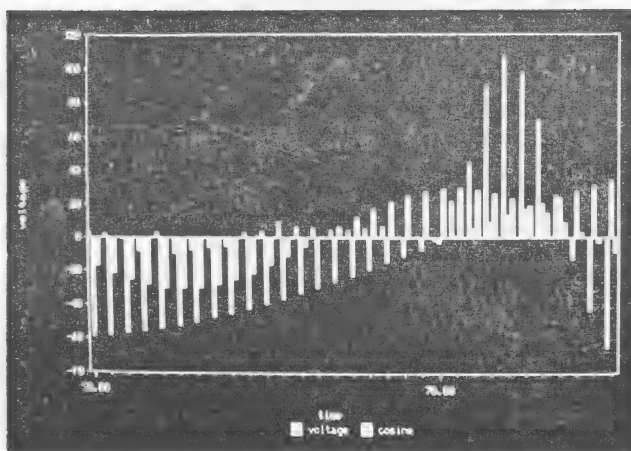
а)



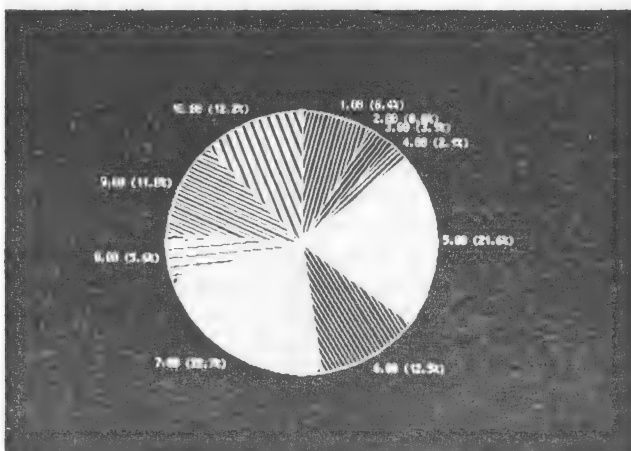
б)



в)



с)



д)

Рис. 2.12. Работа с интерактивным графическим пакетом на ПЭВМ:

- а) главное меню графического пакета;
- б) работа в меню построения графика;
- в) результат построения нескольких графиков с помощью одного из графических пакетов;
- г, д) представление нескольких графиков в режиме гистограммы и с помощью круговой диаграммы

записать все параметры в специальный файл параметров графика, а при входе в пакет указать, что график будет строиться на основе заданного файла параметров.

Мы надеемся, что к этому моменту у читателя сложилось общее представление о возможностях распространенных графических систем ПЭВМ. Опишем некоторую типичную последовательность действий, необходимую для построения графика по данным, записанным в файле:

1) вызовем пакет, например

C: > GRAF

2) на экране возникнет основное меню.

Для примера, на рис. 2.12, а показано главное меню графического пакета. В это меню включены функции построения линейного графика, редактирования двумерных объектов, анализа поверхностей и построения трехмерных объектов;

3) войдем в меню построения линейного графика. (рис. 2.12, б);

4) загрузим файл данных. Для этого выберем соответствующую функцию и зададим имя файла;

5) ответим на вопросы системы о типе шкалы графика, шаге по осям, числе и расположении меток, цвете графика, размере экрана, способе соединения точек, обозначениях осей;

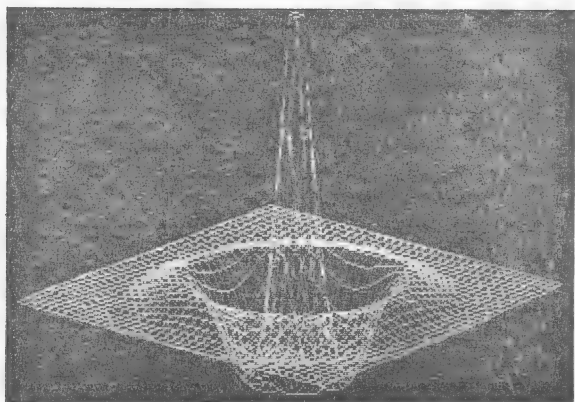
6) построим график с учетом заданных параметров. На рис. 2.12, в, г, д показаны результаты построения графика с помощью соединения точек прямыми линиями, гистограммного графика, приведена круговая диаграмма. Для того чтобы читатель не особенно обольщался красотой приведенных результатов, хочется еще раз напомнить, что большинство интерактивных графических систем предназначены не для работы, а для красивой демонстрации и подготовки данных в отчеты и статьи и важно, чтобы увлечение эстетикой не заслонило основных целей работы на ПЭВМ.

Мы рассмотрели лишь простейшую функцию графической системы: построение графика или семейства графиков. Во многих пакетах предусмотрены также функции построения и редактирования двумерных и трехмерных поверхностей. Естественно, что при увеличении размерности изображаемого объекта возрастает одновременно разнообразие вариантов и сложность реализации соответствующей программы. Для работ, не связанных с анализом пространственных форм деталей, наиболее часто встречается задача построения двумерной поверхности данных, анализе волновых процессов, разработке многомерных фильтров, анализе взаимовлияния параметров в многомерных системах. В простейшем варианте построение сводится к отображению на экране последовательности графиков, соответствующих строкам двумерного объекта. При этом каждая

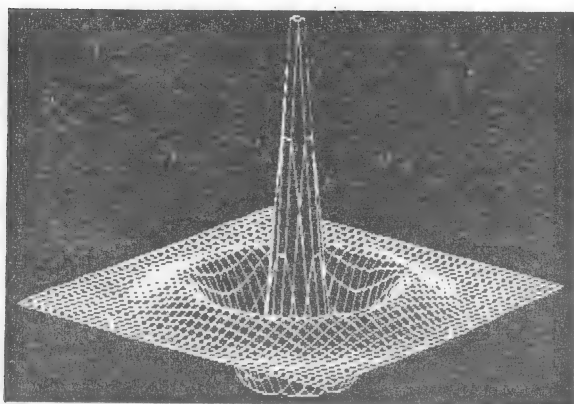
следующая строка для получения перспективы несколько сдвигается относительно предыдущей. Однако при анализе формы сложной поверхности немедленно возникает задача удаления из графика тех частей, которые «заслоняются» участками поверхности, расположенными перед наблюдателем. Это сразу же порождает задачу вращения двухмерного объекта для отображения его скрытых участков. Отметим, что при этом в системе остаются все те же проблемы, связанные с масштабированием графика по осям, задании допустимых значений координат, разметки и множество других. Функции построения поверхностей реализованы в ряде графических пакетов. На рис. 2.13 изображена поверхность, построенная двумя способами: без удаления (а) и с удалением (б) перекрывающихся участков изображения. Графический анализ трехмерных объектов будет рассмотрен в § 2.5, посвященном системам автоматизированного проектирования.

В ряде интерактивных графических систем используются функции аппроксимации экспериментальных кривых. Например, в системе Curve Fitter фирмы Interactive Microwave, пользователю предлагаются различные варианты обработки экспериментальных данных: сглаживание, линейная, сплайн-интерполяция, подбор наилучшего полинома заданной степени и приближение по методу наименьших квадратов. Здесь интерактивность работы начинает играть решающую роль: можно быстро прогнать несколько вариантов приближения экспериментальной кривой и выбрать из них наилучший. Например, на рис. 2.14 показан результат аппроксимации спектра излучения методом сплайн-интерполяции.

Рассмотрим теперь реализацию графических функций в языках программирования и в пакетах прикладных программ. Вообще говоря, минимальный набор графических функций состоит из пяти — шести команд и позволяет рисовать на экране почти все, что угодно. Прагматический и разумный подход к графике реализован в языке Бейсик для ПЭВМ, в котором небольшое число графических функций встроено непосредственно в язык программирования. Но вначале расскажем о возможностях стандартного графического адаптера ПЭВМ. В режиме черно-белого изображения графики можно рисовать с разрешением 640×400 точек. В Бейсике этот режим можно установить, дав команду SCREEN 2. Если дать команду SCREEN 1, то мы сможем работать с разрешением 320×200 точек, но одновременно сможем видеть четыре цвета. Теперь для построения почти всех возможных вариантов графики нам будет достаточно нескольких простых команд. Выбор цветовой гаммы обеспечивается оператором COLOR. Например, дав команду COLOR 0,0, мы выбираем черный фон экрана и нулевую палитру. При этом цвета в этой палитре имеют следующие числовые



a)



б)

Рис. 2.13. Пример построения поверхности с помощью графического пакета:

- a)* без удаления заслоненных участков изображения;
- б)* с удалением заслоненных участков изображения

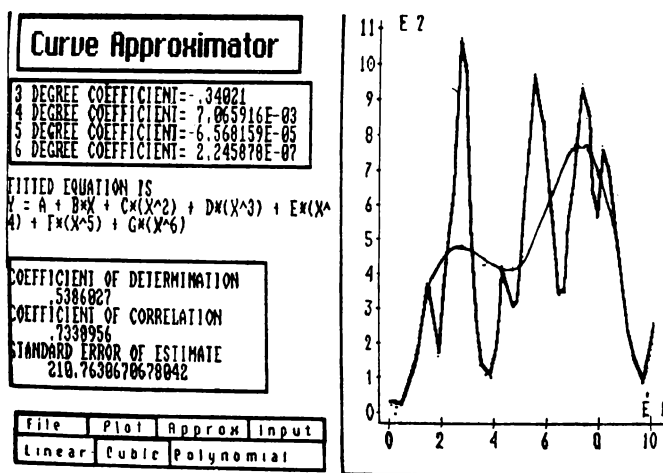


Рис. 2.14. Пример работы с системой аппроксимации графических данных на ПЭВМ: результат аппроксимации исходного графика полиномом заданной степени

коды: 1 — зеленый, 2 — красный и 3 — коричневый. Одна из графических команд — команда LINE. Она записывается следующим образом:

LINE (x1, y1)—(x2, y2), color.

где x1, y1 координаты начальной точки; x2, y2 — координаты конечной точки; color — цвет. Команда LINE соединяет указанные точки прямой линией. Теперь уже можно добиться построения графика на экране. Пусть у нас имеется массив A(100), в котором записаны неотрицательные данные и мы хотим построить график, желательно проведя координатные оси. Для простоты игнорируем разметку осей и начнем строить их из точки с координатами y = 199, x = 0 (самая нижняя точка экрана). Тогда построение зеленых осей выглядит следующим образом:

LINE (0, 199)—(319, 199), 1

LINE (0, 0)—(0, 199), 1

Теперь наша задача — привести значения чисел в массиве A в соответствие с абсолютными координатами точек экрана: Действительно, если максимальное число в массиве A больше чем 200 (максимальный размер экрана), то график просто не поместится на нем. Пусть максимум массива A равен некоторому числу AMAX и мы хотим построить график, занимающий весь экран. Тогда пересчитаем значения массива A следующим образом:

$A(I) = A(I) * 200 / AMAX$

Часть программы, занимающаяся построением графика (пусть он будет красным), будет выглядеть следующим образом:

```
FOR I = 1 TO 99  
  LINE (I, 199 - A(I) * 200 / AMAX) - (I + 1, 199 - A(I + 1) *  
200 / AMAX), 2  
NEXT I
```

График построен, но мы столкнулись с необходимостью пересчета координат, в которых он задан (максимальный индекс 100, максимальное значение AMAX), к координатам графической части экрана. Координаты экрана обычно называются физическими координатами, а реальные значения переменных — мировыми координатами. В Бейсике применяется функция WINDOW, позволяющая производить пересчет этих координат. Некоторые другие важные графические функции Бейсика ПЭВМ будут описаны в § 6.3.

Одним из наиболее распространенных и популярных графических пакетов на ПЭВМ является HALO. Он состоит из нескольких десятков процедур, вызывать которые можно из языков высокого уровня, прежде всего из Си и Фортрана. Рассмотрим основные процедуры пакета.

Несколько процедур обеспечивают пересчет координат графика. Функция setworld (x1, y1, x2, y2) определяет границы мировых координат, в пределах которых возможна работа с графикой. Например, задав функцию setworld (0, 0, 1000, 1000), можно работать с массивами, содержащими до 1000 точек и имеющими максимальное значение до 1000. Гибкое изменение масштаба графиков (режим лупы), естественно, реализуется функцией setwindow (x1, y1, x2, y2). Например, построив график в заданных координатах (0, 0, 1000, 1000) и перестроив его при задании функции setwindow (0, 0, 100, 100), мы увеличим начальный участок графика в 10 раз.

В пакете HALO используется несколько типов курсора: графический (используется при выполнении графических операций), текстовый (используется для вывода текста) и волосяной (используется при работе с точками изображения). Имеются специальные функции для определения положения того или иного курсора и его изменения. Например, установив текстовый курсор в соответствующую позицию изображения, можно специальной командой наложить на изображение текстовую информацию.

В пакете предусмотрены различные функции рисования линий. При этом существует возможность изменять цвет линий и их толщину. Функция setcolor устанавливает цвет, функция setlinewidth — ширину, setlinstyle — тип линии (сплошная, точечная, прерывистая). Для рисования графика предусмотрены специальные функции polylnabs (xarray, yarray, n) и polylnrel (xarray, yarray, n), где xarray — массив значений переменной x; yarray — массив значений переменной y; n — число

точек. Специальные процедуры предназначены для построения окружностей и многоугольников.

Ряд специальных функций предусмотрен для вывода текста на экран. При этом существует возможность изменять размер текстовых символов. Предусмотрены разнообразные функции закрашки областей (сплошным цветом, специальной штриховкой), особые функции необходимы для закрашки сложных многоугольников.

Функции `movefrom` и `moveto` копируют участки видеопамати в массив и могут выводить этот массив в память изображения в различных режимах (выполняя логические операции между фоном и изображением). Такие функции полезны при создании мультипликации.

Один из наиболее мощных графических пакетов для адаптера улучшенной графики ПЭВМ — `EGA ToolKit` содержит несколько десятков процедур, вызываемых с языков Си, Паскаль и Фортран.

Используя графический пакет, можно легко создать свою собственную графическую подсистему, максимально адаптированную под набор конкретных задач, решаемых пользователем. Например, если читатель вспомнит задачи в § 2.1 и 2.2, то он сможет представить себе, как объединить в единой программе вызовы процедур, считывающих значения напряжений с выходов какого-либо прибора, с процедурами обработки сигналов и с функциями построения графиков этих сигналов и их вычисленных параметров. Список задач легко продолжить и, по-видимому, читатель сам сможет придумать задачи, которые он сможет решить.

2.5. СИСТЕМЫ АВТОМАТИЗИРОВАННОГО ПРОЕКТИРОВАНИЯ

Наверное, в памяти каждого выпускника технического вуза живет яркое воспоминание о черчении. Курс черчения необходим, без него конструктор не сможет реализовать свои идеи, но как мала доля творческого труда в этом процессе: представив себе вид разрабатываемого узла, приходится долго и скрупулезно вырисовывать его на листе ватмана, одно неверное движение — и вся работа идет насмарку. Вспомним известный рассказ О'Генри о молодом архитекторе, который пользовался черствыми булочками вместо ластиков. Когда его проект подошел к завершению, сердобольная продавщица пожалела мо-

лодого человека и, разрезав булочки, намазала их маслом, тем самым испортив его многомесячный труд. А чего стоит овладение искусством чертежного шрифта, вырисовывание штампов и спецификаций?

Системы автоматизированного проектирования на базе ПЭВМ в течение ближайших 10—15 лет должны преобразовать работу конструктора. Так, согласно прогнозам, уже к 1990 г. каждый пятый конструктор в мире будет работать на компьютере, а к 2000 г. ручной труд, по-видимому, должен уйти в небытие. Этот процесс не только проявление нежелания мирового сообщества конструкторов стоять у кульмана: уровень сложности современного производства и требования к интенсификации труда уже просто не оставляют перед человечеством другого выхода. Человеку становится не под силу самому произвести разводку сложной печатной платы, он должен уделять свое рабочее время поиску новых решений, т. е. инженерному труду в полном смысле этого слова.

Применение ПЭВМ в качестве помощника инженера-конструктора при подготовке чертежей, составлении спецификаций, разводке печатных плат в настоящее время занимает одно из первых мест в списке областей использования ПК, уступая, быть может, только конторским применениям. Тираж наиболее популярных пакетов автоматизированного проектирования достигает сотен тысяч, многие из них уже стали чем-то вроде международного стандарта. Автоматизированное проектирование развивается уже два десятилетия и активно используется в различных организациях на больших и мини-ЭВМ. За рубежом также получили распространение профессиональные станции для автоматизированного проектирования. Большие ЭВМ и профессиональные станции во многом обгоняют ПК: по скорости расчетов, графическим возможностям, емкости памяти. Однако в большинстве своем они доступны лишь крупным организациям, в то время как появление ПЭВМ дает возможность любому инженеру иметь персональную систему на своем рабочем столе.

Существующие системы автоматизированного проектирования можно условно разделить на несколько категорий и по областям применений, и по уровню возможностей.

По областям применений существует довольно четкое разделение: примерно половина прикладных систем посвящена автоматизации конструкторских работ общего назначения, т. е. интерактивной разработке чертежей различных деталей и конструкций; другая половина систем ориентирована на проектирование и разводку плат и принципиальных схем.

По уровню сложности системы делятся примерно на три категории. Самый простейший уровень — это удобный графический редактор, заменяющий карандаш и ластик, в который встроены некоторые функции, облегчающие конструкторскую

работу. Например, система для разводки печатных плат такого рода позволяет рисовать отверстия, контакты, переходить с одного слоя платы на другой и проводить соединения выбранных пользователем точек линиями заданной ширины таким образом, чтобы избежать пересечений с другими соединениями, наносить на экран текст и обозначения размеров. При этом пользователь сам решает все вопросы размещения элементов и компоновки печатной платы, как и при рисовании ее на листе бумаги.

Следующий уровень сложности — полуавтоматические системы. Пользуясь такими системами, пользователь может извлекать из специальной базы данных стандартные графические изображения деталей или электронных компонентов, проводить процедуры автоматической трассировки, генерировать технические спецификации. Наконец, наиболее развитые и сложные системы включают в себя процедуры моделирования и расчета характеристик разрабатываемого объекта (например, расчет элементов детали на прочность, логическое моделирование работы электронной схемы, построение печатной платы исходя из принципиальной схемы), а также специальные языки управления объектами, с помощью которых пользователь может составлять проблемно-ориентированные приложения.

В целом можно сказать, что авторам еще не приходилось видеть более сложных и объемных систем, разработанных для ПК, чем системы автоматизированного проектирования. Только размер программ и библиотек для некоторых из них составляет десятки мегабайт памяти. Насколько в действительности системы автоматизированного проектирования облегчают конструкторский труд? Рассмотрим модельную ситуацию: имеется опытный конструктор, подготавливающий чертеж вручную, и обученный пользователь системы проектирования, разрабатывающий его на ПЭВМ «с нуля», т. е. не использующий библиотеки стандартных компонентов, а просто рисующий его с помощью курсора и заданных функций. Оставляя в стороне все остальные моменты, можно с уверенностью сказать что с помощью ПЭВМ чертеж будет подготовлен в 2,5—3 раза быстрее. Теперь учтем, что этот чертеж будет выполнен с предельной аккуратностью, может быть в любой момент быстро модифицирован и размножен, что в нем будут соблюдены все ГОСТы, что может быть автоматически подготовлена спецификация, что могут быть эффективно использованы библиотеки стандартных деталей, и т. д. В среднем, эффективность конструкторского труда при использовании систем автоматизированного проектирования повысится в 5—10 раз.

Рассмотрим стандартную периферию ПЭВМ, необходимую для автоматизированного проектирования. В простейшем случае для решения конструкторских задач достаточно стандарт-

ного ПК с графической платой и матричного принтера. При этом распечатки получаемых изображений можно делать на принтере. Однако для повышения разрешения и качества генерируемых изображений в настоящее время все больше используются специализированные графические устройства и графопостроители, обеспечивающие вывод изображения. Графических плат в настоящее время уже выпускается несколько десятков. Среди них наиболее распространены адаптеры улучшенной графики (Enhanced Graphics Adapter), графические платы Galaxy, Artist. Адаптер улучшенной графики заменяет стандартный цветной адаптер ПЭВМ и эмулирует его работу, но с его помощью можно добиться разрешения 640×350 точек с одновременным отображением 16 цветов из палитры 64 цвета. Специализированные платы типа Galaxy требуют дополнительного высококачественного монитора и обеспечивают графическое разрешение до 1024×1024 точки и богатую палитру цветов (до 16 млн.) Кроме того, во многие графические платы встроены специализированные процессоры для выполнения графических операций, например рисования векторов. Это может в десятки и сотни раз увеличить скорость построения сложных изображений. Для управления курсором обычно используется устройство «мышь» или track-ball. В ряде систем автоматизированного проектирования предусмотрена также возможность ввода произвольных графических изображений (в том числе чертежей) в ЭВМ с помощью телекамеры.

Системы автоматизированной разработки чертежей

Перейдем к описанию типичной системы автоматизированного проектирования. Вначале рассмотрим функции системы общего назначения для подготовки произвольных чертежей.

Система автоматизированного проектирования должна решать несколько различных задач: дать пользователю возможность производить геометрические построения, соблюдать стандарты нанесения размеров, штриховок, уметь работать с проекциями изображения, иметь библиотеку стандартных модулей (например, типовых шурупов и резьб).

Типичная система автоматизированного проектирования состоит из набора меню, предоставляющих пользователю различные возможности:

- визуализация и вывод изображения,
- графический редактор,
- работа с трехмерными изображениями,
- измерения на изображениях,
- текстовый редактор,
- работа с библиотекой графических и текстовых объектов,

работа с базой данных объектов,
подготовка технической документации,
моделирование свойств объектов,
программирование функций с помощью встроенного языка,
системные функции.

В развитых системах автоматизированного проектирования каждый из разделов приведенного меню может содержать сотни функций и команд. Описать хотя бы малую часть этих команд на примере конкретной системы не представляется возможным. Поэтому мы попытаемся сделать их обзор «с высоты птичьего полета», а затем проиллюстрируем построение простого чертежа, чтобы у читателя не создалось впечатление, что освоить такую систему очень сложно.

Общие функции визуализации. Во-первых, пользователь может разбить весь чертеж на большое число объектов или слоев (обычно до 1024). Что такое слой? Понятие слоя естественно для задач разводки печатных плат. В простейшем случае разводка на одной стороне платы может быть названа первым слоем, а на противоположной стороне — вторым. Для чертежей произвольных конструкций пользователь может задать части чертежа, имеющие отношение к одной системе, как принадлежащие одному слою. Например, все соединительные элементы можно обозначить принадлежащими одному слою. Другая возможность — рассматривать конкретный объект конструкции, например двигатель, как один слой, а его крепление — как другой. Каждый слой можно раскрасить своим цветом (если, конечно, хватает возможностей используемой графической системы). Такая возможность очень полезна для визуализации отдельных составляющих чертежа. Более того, любой слой или сочетание слоев чертежа можно выключить, включить или заставить мигать. Таким образом, существуют различные варианты визуализации отдельных объектов и элементов конструкции. Можно выбрать такой режим работы, что модификации будет подвергаться только выбранный слой или группа слоев, а остальные элементы чертежа останутся незатронутыми.

Пользователь может одновременно видеть только часть чертежа большой конструкции. При этом он может произвольно смещать окно, через которое он наблюдает чертеж, в пределах заданной области, менять масштаб представления данных.

Линии чертежа могут быть нарисованы различными способами: можно изображать сплошные, штрихпунктирные, штриховые линии. Существует возможность в широких пределах изменять толщину линий.

Каждому объекту изображения пользователь может присвоить свое обозначение. Можно потребовать заштриховать объекты. Тип штриховки может быть самым разнообразным: линиями определенной толщины и ориентации, набором спе-

циальных текстур, соответствующих принятым стандартным обозначениям для различных материалов (стекла, бетона, грунта, песка и т. п.).

Пользователь может стереть любой участок изображения или любой объект: от точки или линии до чертежа достаточно сложного устройства. Любой участок изображения или объект может быть преобразован различными способами: выделенный объект можно сохранить в памяти и перенести в другой участок чертежа или скопировать. Можно произвести параллельный перенос выделенного объекта и повернуть его на заданный угол. Предусмотрена также возможность зеркального отображения или поворота объекта относительно оси симметрии. Существует возможность изменения масштаба объекта или всего чертежа.

Чертеж можно представить в разных системах координат: ортогональной, изометрической, полярной. При этом на экране будут все время отображаться оси соответствующей системы координат, перемещающиеся вместе с курсором, что очень удобно (рис. 2.15). Во многих пакетах автоматизированного проектирования предусмотрена работа с трехмерными объектами. При этом существует ряд возможностей: изменение масштаба по любой из осей, визуализация объекта со скрытыми линиями или без них, вращение получаемого трехмерного объекта, изменение точки зрения и перспективы, наложение теней и штриховка боковых участков трехмерного объекта и т. п. При этом пользователь видит перед собой реальное трехмерное изображение и может двигать его, рассматривать с различных углов зрения, вращать, переворачивать.

В любой момент можно запомнить получаемое изображение и потребовать вывода его на графопостроитель или принтер. Качество чертежа на графопостроителе, как правило, значительно превосходит качество распечатки, получаемой с помощью матричного принтера. В настоящее время распространены компактные и недорогие многоцветные графопостроители. Изображения чертежей, запомненные на дисках, можно передать в другую систему, используя международный стандарт на хранение чертежей IGES (Initial Graphics Exchange System).

Работу с системой автоматизированного проектирования на ПЭВМ можно сравнить с работой художника или фотографа, делающего фотомонтаж. Фактически мы описали функции: раскраски в произвольные цвета, рисования различных частей и накладывания их друг на друга, как будто они нарисованы на прозрачных кальках. Кроме того, можно рассматривать объекты чертежа через увеличивающую или уменьшающую лупу, вырезать их из изображения, как угодно поворачивать, копировать, накладывать, рассматривать с различных углов зрения в перспективе и т. п. Однако эти функции, предоставляемые



a)



б)

Рис 2.15 Последовательность работы в системе автоматизированного построения чертежей на ПЭВМ:
а) экран при начале работы с системой; б) последовательное построение отрезков прямых линий

ПЭВМ, по гибкости и скорости выполнения значительно превосходят все мыслимые возможности человека.

Графический редактор. Графический редактор безусловно является основным элементом системы. В него включается широкий набор функций для вычерчивания на экране самых разнообразных объектов.

Графические примитивы. Для вычерчивания самых разнообразных деталей достаточно автоматической генерации ограниченного набора графических примитивов. Про-

стейшими из них являются точки. Точки можно задавать определенных размеров и цвета. Их расположение на экране в простейшем случае указывается курсором. Существует возможность задания относительных или абсолютных координат точек. При расположении точек на уже нарисованных геометрических объектах предусматривается целый ряд возможностей. Во-первых, точки могут быть заданы в характерных участках чертежа объекта: в самой верхней, нижней, правой или левой границах объекта, в центре тяжести объекта, в средней позиции объекта по выбранному направлению, в центре дуги или окружности, в фокусах эллипсов. Существует также возможность расположения точек на заданном расстоянии от любого выделенного участка чертежа в любом направлении. Кроме того, можно выделить ось симметрии детали и потребовать расположить на этой оси равноудаленный набор точек.

Не меньшее богатство возможностей существует при вычерчивании отрезков прямых линий. Наиболее распространена ситуация, когда пользователь задает начальную и конечную точку и затем дает команду на их соединение. Кроме того, можно провести линию под заданным углом к данной, найти касательную к кривой в заданной точке, провести параллельную линию или набор параллельных линий с заданным пространственным шагом и на заданном расстоянии от исходной. Уже начерченную линию можно разделить на заданное число отрезков, обозначить их координаты и т. п. Можно также потребовать соединения двух заданных линий отрезком дуги заданного радиуса или прямой.

Такой простой объект, как окружность, в развитых пакетах можно задать десятком способов. Простейший и наиболее естественный из них — указать в числовом виде или курсором центр окружности и ее радиус. Другой метод — указать центр и хотя бы одну точку окружности. Третий метод — указать три точки, лежащие на заданной окружности. Четвертый метод — задать окружность, равноудаленную от выделенных точек изображения.

Еще более изощренными являются способы задания дуг окружности. Дугу можно построить, задав: центр и две крайние точки, центральную точку и касательную, любые три точки, через которые проходит дуга (всего встречается до 39 способов задания дуг).

Добавим к этому списку три способа задания треугольников, шесть способов задания прямоугольников, десять способов задания шестиугольников, шесть способов задания эллипсов, девять способов задания многоугольников и т. д. Надеемся, что у читателя еще не пошла кругом голова.

В пакетах автоматизированного проектирования также имеется возможность задания произвольных кривых: гиперболы,

параболы, построения сплайна, проходящего через заданные точки, сегментов перечисленных кривых. Каждый из перечисленных объектов можно частично уничтожить, к каждому можно найти касательную или перпендикуляр и т. д. В целом только для построения простейших графических примитивов таких, как точка, линия, окружность, дуга, эллипс, имеется более сотни команд. В зависимости от выполняемой работы пользователю могут понадобиться различные их сочетания, но, по-видимому, большинство людей, занимающихся автоматизированным проектированием, использует лишь 10—15% всех имеющихся возможностей.

Сопряжения объектов. С помощью уже описанных функций построения графических примитивов можно решить практически все задачи сопряжения кривых. Для большей простоты ряд функций сопряжения обычно выносится в отдельные меню. При этом пользователь указывает, какие объекты он желает сопрячь кривой. Предусмотрены следующие основные варианты: сопряжения двух линий отрезком дуги, сопряжение линий двумя дугами, сопряжения окружностей, сопряжения окружности и прямой и др.

Кроме перечисленных функций, пользователь может в любой момент включать в чертеж стандартные изображения, доступные в имеющейся у него библиотеке объектов. Это могут быть типичные узлы, болты, соединения, модули и т. п. С каждым из этих объектов может быть сделан широкий набор операций: изменение масштаба, повороты, произвольное перенесение из одного участка чертежа в другой.

Работа с трехмерными изображениями. Основные функции, выполняемые в рамках этого подраздела, заключаются в том, что пользователь вначале задает чертежи всех видов объекта. Как только этот этап пройден, система может автоматически сгенерировать трехмерное изображение объекта и представить его самыми различными способами: в изометрии или диметрии, в различном масштабе. Пользователь может выбирать угол зрения на объект, вращать объект, убирать или накладывать скрытые линии и поверхности, затенять или закрашивать любые плоскости объекта. Теперь открывается богатство возможностей: достаточно изменить любой из видов чертежа, как немедленно будут пересчитаны все основные виды и трехмерное изображение объекта. В некоторых системах существует возможность одновременно визуализировать все основные виды и трехмерное изображение объекта. Разработка детали при этом крайне удобна: достаточно задать базовую форму детали, а затем в интерактивном режиме вносить в нее новые элементы, при этом достаточно изменить лишь один из видов и мгновенно получить на экране все изменения в чертежах и в форме разрабатываемой конструкции, да еще и посмотреть на трехмерное изображение под разными углами зрения.

Следующая возможность — автоматическое построение произвольного сечения или разреза. Для этого достаточно указать секущую плоскость на трехмерном объекте и мгновенно получить изображение сечения. В ряде пакетов существует возможность строить сложные сечения, например задавая поверхность, по которой необходимо рассечь трехмерный объект. В получаемом сечении можно даже автоматически произвести всю необходимую штриховку.

Работа с трехмерной графикой в системе автоматизированного проектирования не только развивает пространственное воображение, но и значительно повышает эффективность конструкторского труда, так как у инженера появляется мощный инструмент манипулирования с пространственными формами. Если описанный выше графический редактор можно назвать редактором чертежей, то в данном случае мы имеем дело с редактором поверхностей. В пакеты для проектирования трехмерных объектов включены самые разнообразные функции построения поверхностей: сфер, эллипсоидов, плавного сочленения двух плоскостей и т. п. Например, систему можно попросить соединить две плоскости, произвольным образом ориентированные в пространстве, гладкой поверхностью, потребовать, чтобы эта поверхность проходила через заданные точки. Читатель может себе легко представить возможное число процедур редактирования трехмерных изображений, если учесть их число для двумерного случая.

Измерения на изображении. Основной целью этой подсистемы является автоматическое измерение и простановка размеров на чертеже. Для того, чтобы что-то измерять, вначале необходимо задать абсолютные размеры всего чертежа. Обычно в пакетах практически не существует ограничений на числовые значения этих размеров — они могут иметь диапазон от микронов до километров. При нанесении размеров пользователю предоставляется большое число вариантов. Простейший способ измерений — интерактивный. При этом необходимо указать крайние точки объекта измерения и дать запрос на численное значение. Для облегчения этого процесса предусмотрен набор измерительных функций, например пользователь указывает курсором на окружность и дает запрос: «Измерить радиус окружности». Среди простейших и стандартных типов измерений размера можно также выделить размер по горизонтали, размер по вертикали, наименьшее расстояние между объектами, расстояние по перпендикуляру от заданной прямой до выделенного объекта, угол, длина дуги.

Кроме интерактивного обычно предусмотрен и автоматический способ измерения размеров. При этом пользователь может указать список выделенных объектов, которые необходимо измерить, и тип измеряемого параметра, а система автоматически рассчитает размеры (а иногда и сама проставит все необхо-

димые обозначения: линии, стрелки, числа и т. д.). Вообще, процедура расстановки размеров на чертеже — одна из самых сложных. Для того чтобы в этом убедиться, достаточно обратиться к справочнику по черчению: необходимо определить ориентацию надписи, ее расположение, для различных объектов существуют различные правила нанесения размеров. В ряде пакетов автоматизированного проектирования эти правила учтены. Более того, учитываются стандартные правила нанесения размеров на деталях, хранящихся в библиотеке стандартных модулей. Естественно, что все размеры автоматически пересчитываются при изменении масштаба изображения. Кроме стандартных размеров, пользователь имеет возможность измерения и более сложных геометрических характеристик объектов, например площади поверхности, периметра, кривизны, первых и вторых моментов инерции относительно заданных осей. Для трехмерных объектов также можно вычислить поверхности, объемы, объемы участков между выделенными элементами поверхности, координаты центров тяжести и др.

Обычно пользователю предлагаются различные варианты расположения размерных линий и текста на экране, множество типов стрелок, ориентация текста, центровка размера относительно выделенного участка изображения и т. п. Эти возможности позволяют достаточно быстро подготовить чертеж.

Текстовый редактор. Предназначен для нанесения на чертеж текста и комментариев. Естественно, что при этом предусмотрено множество сервисных функций: набор стандартных шрифтов, цвет текста, его толщина, высота и ориентация. Предусматриваются также сервисные функции типа расположения надписи по центру между выделенными участками чертежа, задание интервалов между буквами, исправления ошибок, замены символов или их сочетаний на заданные и т. п.

Работа с библиотекой графических и текстовых объектов. Мы уже упоминали, что любой созданный объект можно назвать каким-то именем и запомнить в специальной библиотеке. Кроме того, существуют стандартные библиотеки объектов: соединительных устройств, болтов, гаек, втулок и чего угодно в зависимости от области применения системы. При этом пользователю предоставляются возможности: просмотреть имеющуюся библиотеку, считать из нее и записать в нее любой символ или объект с сопровождающей его информацией: размерами, названием, комплектующими устройствами и др. (подробнее см. п. «Работа с базой данных»). При работе с библиотекой стандартных модулей можно произвести ряд операций. Во-первых, можно изменить масштаб и ориентацию объекта, скопировать объект и поместить его в любое место на чертеже. Во-вторых, можно в отдельное окно экрана прочитать изображение нескольких объектов из библиотеки (например, чертежи

двух стандартных деталей, соединяющихся шурупом, и шурупа), скомпоновать их и записать в библиотеку под новым именем. Во многих пакетах существует возможность объединения нескольких десятков объектов. Описанная возможность очень важна при использовании пакета для конкретных приложений, когда в распоряжении пользователя всегда имеется набор стандартных модулей, из которых он может компоновать новое изделие. Естественно, что существуют и библиотеки трехмерных модулей, которые можно «состыковать» друг с другом.

Работа с базой данных объектов. Читателя, интересующегося принципами работы с базами данных, мы отсылаем к § 2.7. Здесь же рассмотрим только идеологию использования баз данных в системах автоматизированного построения чертежей. Для каждого из объектов системы пользователь может составить его описание, включающее различные сведения: наименование объекта, условное обозначение, порядковый номер, тип устройства, составной частью которого является данный объект, материал, детали, составляющие объект, их количество, стандартные изделия, входящие в объект, их количество и условные обозначения, типы соединений, объекты, соединяемые с данным, массу, объем, площадь, стоимость, механические характеристики и т. д. Пользователь может сам задать те сведения, которые считает необходимыми. Для этого существует возможность формирования типа записи информации об объекте (см. § 2.7). На экране при этом возникает меню, в котором изображены колонки текста, упорядоченные по элементам описания, например:

Наименование подсистемы:	двигатель
Наименование объекта:	гайка
Тип:	нормальной точности
Диаметр резьбы:	8 мм
Шаг резьбы:	1,25 мм
Размер под ключ:	13 мм
Высота:	6,5 мм
Количество:	45 штук
Масса:	10 грамм
и т. д.	

Для объектов, записанных в стандартных библиотеках, такое описание, как правило, уже имеется. Таким образом, в базе данных объектов хранится вся возможная информация об элементах конструкции.

Используя информацию, описывающую чертежи и входящие в них объекты, пользователь получает чрезвычайно широкие возможности. Он может формировать самые разнообразные запросы к базе данных. Для того чтобы читатель понял характер этих запросов, приведем несколько простых примеров. Предположим, что подготовлен чертеж какого-нибудь сложного

устройства. Теперь можно легко сформировать и получить ответ на следующие вопросы:

сколько объектов используется в данной конструкции, сколько винтовых соединений применено в конструкции, сколько в данной конструкции элементов, сделанных из стали определенной марки, какова их масса и общая стоимость, какую долю объема конструкции занимают объекты, относящиеся к заданной системе (например, к системе охлаждения двигателя) и какова их общая площадь.

Список вопросов можно продолжать почти до бесконечности. Читатель, видимо, понял, что теперь в руки конструктора попал мощный инструмент анализа инженерных решений. Учитывая, что в функциях управления базой данных предусмотрена возможность производить сложные вычисления по формулам, пользователь может немедленно рассчитать технические характеристики разрабатываемого устройства.

Другой аспект применения базы данных — поиск в стандартных библиотеках объектов, которые могут подойти для применения в разрабатываемом изделии, например, какие имеются стандартные зубчатые колеса, изготовленные из стали заданной марки с определенной высотой зубьев.

Наконец, еще одно применение базы данных даже во многом перекрывает уже описанные: выбирая из описания объектов необходимые параметры, можно мгновенно генерировать техническую документацию.

Подготовка технической документации. Эта подсистема основана на работе с базой данных, но обычно выделяется в специальный раздел меню, так как при подготовке спецификации основную роль играет соблюдение текстовых стандартов. Здесь пользователю предоставляется возможность сгенерировать вид текстовой таблицы спецификации и указать, какие именно данные необходимо записать в выбранные места таблицы. Так, согласно ГОСТу спецификация чертежа должна быть выполнена на листах определенного формата и содержать разделы «Документация», «Комплексы», «Сборочные единицы», «Детали», «Стандартные изделия», «Прочие изделия», «Материалы», «Комплекты». Пользователю теперь достаточно один раз проделать процедуру, в ходе которой он установит соответствие между полями базы данных и элементами таблицы спецификации.

Так, для раздела спецификации «Детали» системе генерации спецификации необходимо указать, что в колонку «формат» следует поместить данные из поля «формат» базы данных, в колонку «порядковый номер» поместить значение поля «порядковый номер», в колонку «обозначение» поместить символьную переменную из поля «обозначение» и т. д. Теперь достаточно указать системе, какой именно чертеж необходимо обработать, и будет получена полная спецификация! Внося любые измене-

ния в конструкцию, мы немедленно получаем скорректированную спецификацию. В рамках одной системы теперь можно создать целый ряд специальных форм технической документации и буквально завалить ею всех нуждающихся, если только хватит бумаги.

Моделирование свойств объектов. Любое моделирование подразумевает, что в системе заложена некоторая математическая модель: формулы, по которым рассчитывается прочность детали или мощность тепловыделения, параметры жесткости конструкции и т. д. Различные модели столь разнообразны и настолько зависят от конкретной области применения, что перечислять их здесь не представляется возможным. К наиболее популярным пакетам автоматизированного проектирования обычно прилагаются специальные библиотеки моделей, ориентированных на различные области применения. Например, для механических конструкций, используя такую модель, пользователь может указать внешние силы, приложенные к конструкции, и рассчитать значения напряжений, возникающих в ее различных участках. Подробнее о моделировании речь пойдет в § 2.6.

Программирование функций с помощью встроенного языка. В системе автоматизированного проектирования кроме набора меню обычно существует и набор специальных команд с аргументами, соответствующих выполнению всех операций. Например, для увеличения изображения можно выбрать страницу меню визуализации и функцию «лупа», а можно набрать на клавиатуре команду «ZOOM 3», что означает увеличение изображения в три раза. Число таких команд для развитых систем достигает нескольких сотен. Естественно, что желательно иметь возможность не набирать каждый раз какие-то команды на клавиатуре, а включать их как вызовы стандартных функций в язык программирования. Зачем это может понадобиться? Например, какую-то длинную последовательность стандартных операций необходимо выполнять достаточно часто. Тогда, если мы напишем программу, включающую вызовы необходимых функций, то всю эту стандартную последовательность можно будет выполнить нажатием одной кнопки. Кроме того, в большинстве случаев у таких последовательностей процедур имеются численные аргументы, которые вводятся пользователем или рассчитываются по ходу выполнения программы.

В еще более сложном случае, проводя операции проектирования, необходимо производить сложные математические расчеты. Типичные примеры таких расчетов: анализ разрабатываемой детали на прочность по заданным формулам, приближение поверхности методом конечных элементов, оптимальное размещение компонентов на поверхности изделия. Так, конст-

руктор, разрабатывающий некоторый узел, может составить программу, которая будет пытаться минимизировать площадь объекта исходя из того, что детали не должны перекрываться в пространстве, заданные узлы не должны быть расположены ближе, чем задано, и т. д. Естественным для такой программы является ее объединение с языком данной системы автоматизированного проектирования. Например, программа может обращаться к базе данных, рассчитывать оптимальное пространственное расположение деталей и вызывать процедуры, которые модифицируют имеющийся на экране чертеж с учетом сделанных изменений.

Таким образом, возникает задача синтеза языка программирования с системой автоматизированного проектирования. Наиболее разумным путем решения этой проблемы является непосредственное включение языка в данную систему. Так, в систему AUTOCAD включен язык AUTOLISP — одна из версий Лиспа. Сейчас начинают появляться даже более сложные системы, включающие не только язык программирования, но и экспертные системы для принятия решений и подсказок конструктору в процессе разработки. Например, выпускаются экспертные надстройки над системой автоматизированного проектирования, ориентированные на различные конкретные задачи: проектирование электрических двигателей, разработку несущих конструкций в архитектурных сооружениях и т. д. В эти надстройки включен набор правил и математических моделей и конструктор в процессе работы может получить «советы» по оптимальному выбору тех или иных параметров разрабатываемой системы.

Системные функции. Обычно в системы автоматизированного проектирования включаются стандартные системные функции: просмотреть каталог файлов, задать имя текущего файла, с которым будет работать система, создать конфигурацию экрана, указать системе, какие периферийные устройства будут использоваться для вывода чертежей, в каком стандарте запоминать данные и т. п. В большинстве систем автоматизированного проектирования включены встроенные подсказки, объясняющие пользователю, как работает та или иная команда, что ему необходимо сделать, чтобы добиться того или иного результата.

Будем считать, что к этому моменту читатель представил себе основные функции системы автоматизированного проектирования чертежей на ПЭВМ и понял, что создание такой системы требует длительных усилий большого коллектива людей. Надо сказать, что первые системы такого рода для больших ЭВМ разрабатывались десятилетиями. Для ПЭВМ время разработки аналогичных пакетов сократилось до года. Какие же

пакеты наиболее доступны для ПЭВМ, совместимых с моделью IBM PC? Прежде всего, это пакет AutoCad фирмы AutoDesk, ставший уже международным стандартом. С момента появления этого пакета прошло уже несколько лет, регулярно появляются новые версии. Пакет может работать с трехмерными объектами, включает в себя язык программирования. Вторым по популярности является пакет CADKEY фирмы Micro Control Systems. Известны также пакеты 3D Conception, Anvil 1000, Micro CAD, Solid Modelling и др.

Рассмотрим типичную последовательность работы с системой класса AutoCad при построении простого чертежа.

1. Вызываем программу, для этого набираем на клавиатуре название системы ACAD и даем возврат каретки.

2. На экране возникает главное меню, из которого выбираем функцию «Графический редактор».

3. Теперь мы оказались в странице графического редактирования изображения. Основная часть экрана сейчас пустая (см. рис. 2.15, а), справа видна часть меню, в которой высвечены основные функции, снизу имеется строка команд. В этом режиме графический редактор ждет наших команд. На экране видно перекрестие, соответствующее текущему положению курсора. Линии перекрестия параллельны осям координат системы и передвигаются вместе с курсором.

4. Теперь с помощью «мыши» или клавишами управления курсором мы можем передвигать его по изображению. Начнем с построения прямой линии. Для этого выберем команду LINE. В командной строке появится подсказка: «Введите первую точку». Теперь мы можем, передвигая курсор, указать первую точку и нажать на возврат каретки или на специальную кнопку, расположенную на мыши. Теперь появляется подсказка «Введите последнюю точку» и, выполняя те же операции, мы строим линию. Заметим, что при перемещении курсора на экране появляется так называемая «резиновая» линия (рис. 2.15, б), в реальном времени соединяющая начальную точку линии и текущую точку, в которой расположен курсор. Это позволяет пользователю видеть, какая именно линия будет построена при выборе текущей точки экрана в качестве последней точки линии.

5. Пусть, дойдя до края детали, мы хотим провести дугу. Для этого вызываем команду ARC. Редактор просит: «Введите первую точку», «Введите центральную точку» и «Введите последнюю точку». Выбирая эти точки с помощью курсора, мы строим дугу окружности.

6. Продолжаем наши построения. Предположим, что возникла необходимость сопряжения двух прямых линий. Для этого

даем команду **FILLET**. Редактор вначале запрашивает радиус кривизны сопряжения. Этот радиус легко задать с помощью курсора, установив его на участок сопряжения. Затем система просит указать ей две линии, которые необходимо сопрячь, и выполняет эту процедуру.

7. Пусть внешняя граница детали проведена. Начертим в середине детали отверстия. Для этого вызовем команду **CIRCLE**. Система попросит указать ей центральную точку и радиус и окружность построена.

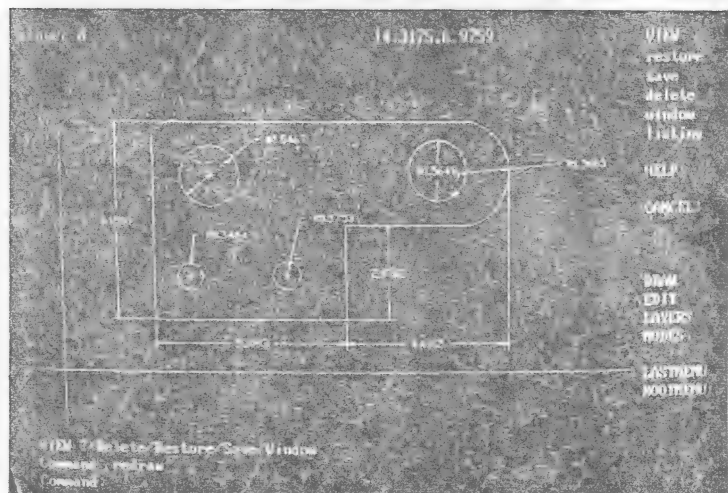
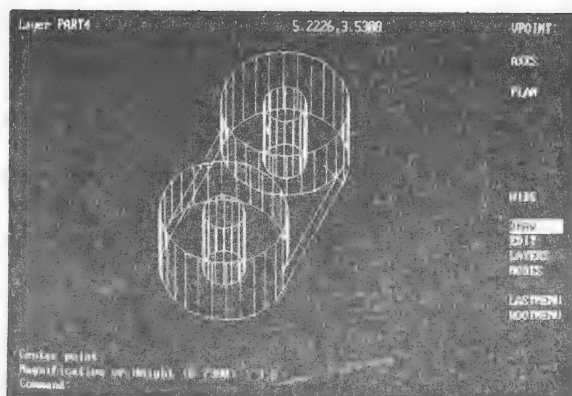


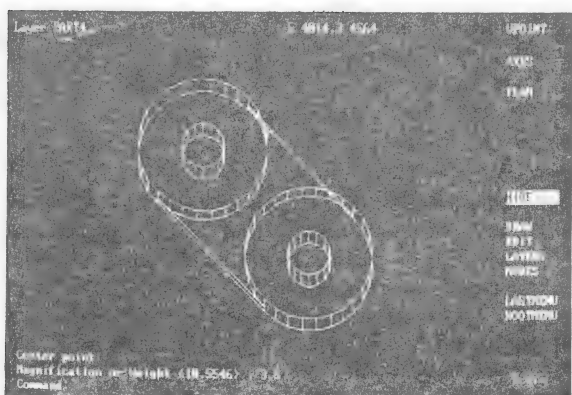
Рис. 2.16. Пример простого чертежа, подготовленного с помощью системы автоматизированного проектирования

8. Проставим размеры. Для этого выберем в меню специальную команду **DIM**. Потребуем проставить размер радиуса окружности. Для этого выбираем из меню пункт **RADIUS** и указываем курсором на окружность или дугу. Система попросит указать ей точку экрана, в которой необходимо написать текст и автоматически проставит размерные линии, стрелки и численное значение радиуса. При простановке горизонтальных или вертикальных размеров система попросит указать точки, из которых необходимо начать проводить размерные линии и автоматически сделает все остальное.

Базовый чертеж готов. Конечный результат показан на рис. 2.16, а. Рис. 2.17 иллюстрирует объемный чертеж.



а)



б)

Рис. 2.17. Построение объемной конструкции в системе автоматизированного проектирования под разными углами зрения

Системы автоматизированного проектирования печатных плат и принципиальных схем

Число различных пакетов для автоматизации проектирования печатных плат в настоящее время в несколько раз превышает число пакетов для разработки произвольных чертежей. Это связано с тремя обстоятельствами: увеличивающейся долей сложного электронного оборудования в промышленном производстве, значительно большей сложностью конструирования и разработки, относительно большей простотой формализации задачи.

Структура пакетов для автоматизации проектирования печатных плат в целом близка к описанной выше структуре пакетов для построения чертежей. Наиболее развитые системы включают в себя следующие основные элементы:

- графический редактор печатной платы,
- графический редактор принципиальной схемы,
- процедуры автоматической трассировки и размещения,
- текстовый редактор,
- визуализация и вывод изображений,
- работа с библиотекой модулей,
- работа с базой данных,
- моделирование работы принципиальной схемы.

Рассмотрим кратко основные функции этих подсистем.

Графический редактор печатной платы. Эта подсистема напоминает редактор изображения. Как и при построении чертежей, пользователь может разбить чертеж на множество слоев, соответствующих слоям или сторонам печатной платы, или, например, выделить отдельным слоем шины питания. Каждый слой может иметь свой цвет, пользователь может выделить для визуализации только один слой или группу слоев. Пользователь обычно видит на экране только часть разрабатываемой им печатной платы. Он может сдвигать графическое окно, наблюдая различные участки изображения, изменять масштаб визуализации.

В распоряжении пользователя имеется библиотека графических изображений стандартных деталей: микросхем, резисторов, конденсаторов и т. д. При этом соблюдены стандартные размеры этих деталей в зависимости от масштаба чертежа, числа выводов. В любой момент разработчик может считать из библиотеки изображения этих деталей и наложить их на изображение, скопировать, повернуть, передвинуть в любой участок печатной платы. При этом существует приятная возможность: разработчик может скомпоновать на поверхности платы основные электронные компоненты, а затем одной командой сформировать изображения контактных площадок для каждого из стандартных выводов деталей.

В памяти графического редактора имеется набор графических примитивов, необходимый для системы. Для системы проектирования печатных плат этих примитивов довольно мало: изображения разъемов, контактных площадок, отверстий, набор соединительных линий стандартной ширины. При этом, как правило, соблюдены стандартные размеры, толщины, но пользователю предоставляется возможность задать и свои собственные параметры.

Пользователь может перемещать курсор по экрану с заданным шагом в миллиметрах. Это очень удобно, например для задачи обозначения выводов микросхемы: установив курсор

на исходную позицию и задав шаг его перемещения, можно быстро нарисовать последовательность контактных площадок для выводов микросхемы.

Одной из основных операций, выполняемых с помощью графического редактора, является соединение различных участков печатной платы проводящими линиями. При этом возможно несколько режимов работы. Простейший из них — разводка платы «вручную». При этом пользователь с помощью курсора отмечает первую точку, а затем двигается по печатной плате и дает команды на соединение выделенных точек отрезками линий. Второй способ — полуавтоматическое соединение. При этом редактору указывается исходная точка, конечная точка и дополнительно одна или несколько промежуточных точек, через которые должно пройти соединение. Это очень удобно, так как разработчик может иметь свои соображения о том, как провести соединение, например обогнуть выделенный участок печатной платы. Наконец, при автоматическом соединении указываются исходная и конечная точки. Редактор при этом «понимает», что он имеет дело с проводящими соединениями и ищет кратчайший путь между указанными точками, одновременно не допуская пересечений проводящих соединений и пытаясь соблюсти технологические допуски на минимальное расстояние между проводниками.

Когда все основные соединения проведены, пользователь может решить видоизменить плату. В графических редакторах для этого предусмотрены различные команды, позволяющие пользователю выделить участок платы и попросить систему перестроить соединения таким образом, чтобы освободить данный участок. Существует также возможность проведения набора параллельных соединений, растяжение чертежа по осям и т. д.

Описанный графический редактор может использоваться в двух основных режимах. Первый из них — подготовка печатной платы «с нуля». При этом редактор выступает в роли помощника разработчика, понимающего, что такое проводящие соединения, и имеющего в памяти стандартные графические обозначения основных компонентов. Второй режим — изменение уже готового изделия, например полученного с помощью автоматических процедур.

Графический редактор принципиальной схемы. В типичной системе такой редактор представляет собой средство для размещения на экране набора стандартных изображений электронных схем и соединения их выводов. При этом пользователь может считать из библиотеки принципиальные обозначения различных элементов (микросхем, транзисторов, конденсаторов и др.), групп элементов (например, стандартных схем), скомпоновать их на экране и соединить их выводы. Работа программ ввода данных о принципиальной схеме по своей

структуре очень близка к системам редактирования, входящих в пакеты для разводки печатных плат. Обычно при этом в специальном окне экрана изображаются условные обозначения элементов и пользователю достаточно указать курсором на элемент, который он желает разместить на принципиальной схеме, а затем переместить курсор на то место экрана, в которое необходимо поместить данный элемент. Естественно, что элемент можно передвигать по рабочей области экрана, поворачивать и т. п. Для каждого из введенных элементов существует библиотека их номиналов, марок (например, типов транзисторов). При этом разработчик может указать один из библиотечных типов либо задать свой номинал (например, указать, что данное сопротивление имеет номинал 10 Ом).

Специальной командой пользователь может проводить электрические соединения. Для этого достаточно указать курсором начальную и конечную точки и дать команду на проведение электрического соединения. Система при этом проведет линию таким образом, чтобы не пересекать условные изображения других деталей. Редактор принципиальных схем включает в себя множество процедур для облегчения проведения соединений. Предусмотрены также различные процедуры для проведения набора параллельных соединений, обозначения переходов и пересечений проводников. Скомпонованная схема может быть сохранена в памяти, причем в отдельном файле запоминается таблица соединений. Разработчик может запоминать любой фрагмент схемы в специальной библиотеке для повторного использования, редактировать уже имеющуюся принципиальную схему и т. п. Читатель, который прочел параграф, посвященный системам автоматизированного проектирования, поймет, что здесь мы имеем дело просто с еще одной версией графического редактора, ориентированного на конкретную задачу разработки и имеющего типичную для таких систем структуру. На рис. 2.18 показан экран ПЭВМ при работе с редактором принципиальной схемы: в нижней строке размещены изображения стандартных электронных компонентов, справа — область меню. Работая с системой, пользователь выбирает стандартные детали и с помощью курсора размещает их в графическом окне, а затем дает команды на соединения выводов.

Процедуры автоматического размещения и трассировки. Эти процедуры достаточно сложны и, как правило, скрыты от пользователя. Они могут быть использованы в двух режимах. Первый из них — когда разработчик, уже подготовивший печатную плату, хочет оптимизировать ее параметры: добиться наиболее плотной компоновки электронных компонентов. Второй — когда эти процедуры используются для автоматической разработки печатной платы.

В первом случае исходной информацией является готовая и разведенная печатная плата. При этом система пытается

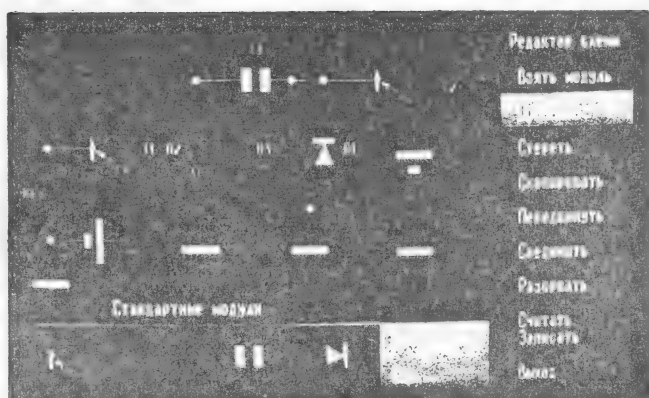


Рис. 2.18. Экран ПЭВМ при работе с редактором принципиальной электрической схемы

модифицировать размещение деталей на плате, сохранив таблицу соединений. Во втором случае исходной информацией является принципиальная схема. Исходя из принципиальной схемы, система находит информацию о библиотеке стандартных изображений электронных компонентов, использованных в схеме, о таблице электрических соединений и начинает попытки оптимальной упаковки компонентов, пытаясь произвести разводку получаемой платы.

Оптимальное размещение компонентов достигается исходя из получения максимальной плотности с учетом технологических допусков и дополнительных требований пользователя. Так, в некоторых пакетах при вызове процедуры размещения можно потребовать не занимать какой-либо участок платы. При выполнении автоматической трассировки система пытается найти оптимальные соединения компонентов, соблюдая допустимые расстояния между проводниками.

Автоматическое размещение и трассировка в идеальном случае позволяют получить готовую печатную плату. Однако эти процедуры весьма сложны, длительны и не всегда могут найти решение задачи с учетом заданных ограничений. Время выполнения операций размещения и трассировки в зависимости от уровня сложности платы может занимать от десятков секунд до десятков минут. Во многие системы включены запросы к разработчику. При этом процедура автоматической трассировки происходит в интерактивном режиме. Например, система может запросить у разработчика разрешение перейти на другой слой платы. В более развитых пакетах система «понимает» характер преобразований в принципиальной схеме. Например,

пусть в какой-то микросхеме используется простая операция типа логической инверсии, а в другой микросхеме имеются свободные выводы, которые могут быть использованы для той же цели. Тогда система может запросить пользователя о возможности видоизменения исходной схемы, так как разводка во втором случае может оказаться проще.

После выполнения процедур автоматического размещения система выдает на экране результат и полученные характеристики: плотность компоновки деталей, площадь свободных участков, минимальный зазор между проводниками и т. д.

Текстовый редактор. Функции текстового редактора практически не отличаются от описанных ранее. Точно так же с его помощью можно нанести на чертеж текст произвольного цвета, размера и ориентации, исправить ошибки текста и т. д.

Визуализация и вывод изображения. Эти функции также типичны. С их помощью можно просмотреть различные слои чертежа, изменить масштаб и цвета, дать команду на вывод изображения на матричный принтер или графопостроитель. При этом большое внимание уделяется соблюдению абсолютных масштабов чертежа в соответствии с принятыми стандартами.

Работа с библиотекой модулей. В системе предусмотрены две различные библиотеки: обозначений элементов на принципиальных схемах и графических изображений деталей. При этом в описание объектов входит большая дополнительная информация. Так, для изображения микросхемы на принципиальной схеме необходимо выводить на чертеж нумерацию выводов и обозначение типа микросхемы, при выводе графического изображения корпуса этой микросхемы необходимо также помнить нумерацию и расположение выводов и абсолютные размеры ее корпуса. Учитывая разнообразие и размеры справочников по электронным приборам и микросхемам, можно представить себе объем подобных библиотек. Кроме того, в библиотеках существует возможность ввести новые приборы или запомнить любые фрагменты принципиальной схемы или печатной платы со всеми имеющимися соединениями.

Работа с базой данных. Читатель, видимо, уже понял, что для того, чтобы обеспечить взаимодействие различных подсистем, работающих в системе автоматизации проектирования, необходимо пользоваться базой данных. Так, для того, чтобы произвести автоматическую разводку печатной платы исходя из принципиальной схемы, система должна найти ссылки на все электрические соединения в данной схеме, на все электронные компоненты, на их размеры, форму, электрические характеристики, на используемые стандарты толщины проводников, зазоров между проводящими соединениями и т. д. Вся эта огромная работа обычно скрыта от пользователя и протекает «за кадром». В то же время пользователь может в любой

момент получить всю интересующую его информацию. Для этого в системе выделена отдельная страница меню работы с базой данных. С помощью этой системы пользователь может получить ответы на различные вопросы, например:

с какой точкой принципиальной схемы соединен третий вывод заданной микросхемы?

сколько в системе используется резисторов сопротивлением 10 Ом и где они расположены?

какие микросхемы в библиотеке модулей имеют заданный размер, тепловыделение и содержат не менее трех инверторов?

какое общее тепловыделение разработанной платы?

Запросы к базе данных можно условно разделить на запросы о текущем проекте (БД проекта), запросы о технологических параметрах применяемых изделий (справочная БД) и запросы о ранее разработанных схемах и узлах (архивная БД). Функции работы с базой данных в ряде систем имеют графический вывод. Так, запросив информацию о расположении заданной детали на печатной плате, ее можно выделить красным цветом, прослеживаемое соединение можно заставить мигать.

При занесении в библиотеку модулей новых компонентов пользователь имеет возможность заполнить специальную таблицу, в которой описаны их параметры. Эта таблица затем используется базой данных.

Вообще, читатель уже заметил, что, несмотря на то, что изменилась область применения САПР, приблизительная структура системы базы данных практически не изменилась.

Как и в системах автоматизации изготовления чертежей, база данных может быть использована для подготовки технической документации. Для этого пользователь может установить соответствие между специальными текстовыми таблицами и элементами записей базы данных и сгенерировать техническую документацию.

Моделирование работы электронной схемы. Эта подсистема является одной из самых нетривиальных. Основные принципы работы пакетов для моделирования схем будут рассмотрены нами в отдельном параграфе. Как правило, моделирование радиотехнических схем реализуется с помощью отдельных пакетов программ, а в системы разработки печатных плат включаются процедуры так называемого логического моделирования. Эти функции кратко рассмотрены в § 2.6.

Какие наиболее распространенные пакеты прикладных программ используются на ПЭВМ для проектирования и разработки печатных плат и принципиальных схем?

Наиболее простая и распространенная система такого рода, известная авторам — пакет SMARTWORK фирмы Wintek. Эта система фактически представляет собой простой графический редактор для подготовки печатных плат. Одна из самых

мощных систем проектирования для IBM PC — пакет PCad фирмы Personal CAD Systems. Этот пакет примерно соответствует приведенному нами описанию и имеет редакторы печатных плат, принципиальных схем, процедуры автоматической трассировки и размещения, функции логического моделирования, базу данных, библиотеки. Пакет PCad позволяет проектировать печатные платы, имеющие до 500 элементов и 2000 связей. Широкую известность получил пакет RedCad фирмы Racal-Redac, имеющий приблизительно те же характеристики, что и пакет PCad. Известны также пакеты CADdy (CADdy Corporation), CT-1000 (Case Technology), Micrograph (Micrograph Systems), отечественная разработка «Магистр» для ЭВМ ЕС1840 и множество других.

Рассмотрим простой пример разводки печатной платы на примере класса SMARTWORK. В этой системе используется всего несколько клавиш: можно управлять курсором, наносить на изображение обозначение контактной площадки (функциональная клавиша F3), проводить соединения (функциональные клавиши F1, F2, F8), изменения слоев чертежа (клавиша PgUp и PgDn). Для того чтобы начать разводку печатной платы, необходимо выполнить приблизительно такую последовательность операций:

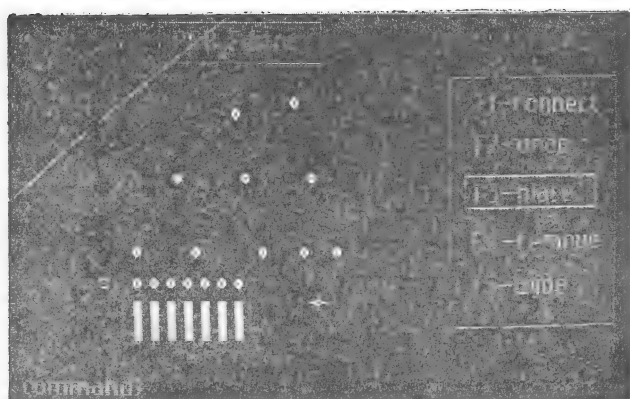
1) загрузим систему, для этого наберем на клавиатуре EDIT и дадим возврат каретки;

2) перед нами появится поле печатной платы. Справа расположено окно подсказок (рис. 2.19, а). Передвигая курсор, выберем точки расположения контактов и поставим их, нажимая на клавишу F3;

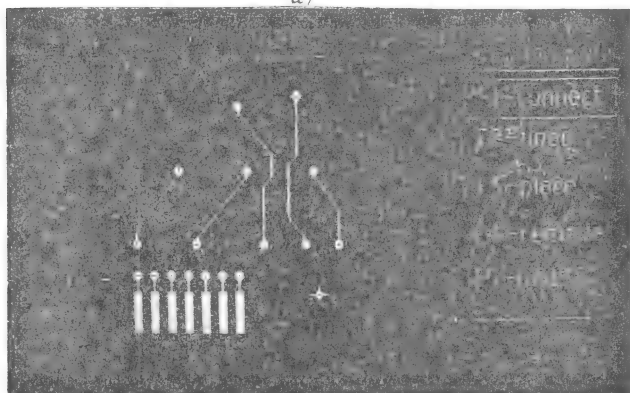
3) начнем процедуру соединения. Для автоматического соединения достаточно отметить две точки клавишей F1. Если мы хотим провести соединение через заданную точку, нажмем F1 несколько раз, пройдя последовательно через промежуточную точку к конечной (рис. 2.19, б);

4) при необходимости сменим слой чертежа, сотрем последнее соединение (клавиша F2), расширим или сузим размер линии. На рис. 2.19, в приведен пример чертежа печатной платы, подготовленного с помощью описанной системы.

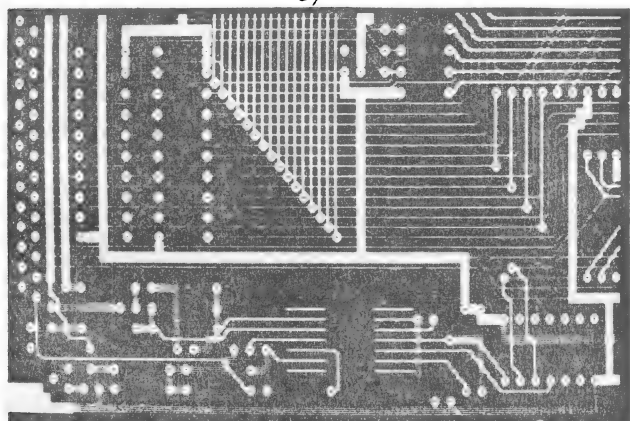
Завершая этот более чем скромный обзор систем автоматизированного проектирования на ПЭВМ, хотелось бы сделать следующее замечание: при освоении подобных систем у пользователей часто возникает психологический барьер, связанный с огромным разнообразием предоставляемых возможностей. Надо сказать, что этот факт имеет под собой основания: научиться работать на подобной системе гораздо сложнее, чем, скажем, выучить команды ДОС. Приведенный нами пример построения чертежа показывает, что все базовые операции крайне просты и естественны. Этих операций, однако, слишком



а)



б)



в)

Рис. 2.19. Последовательность разводки простой печатной платы на ПЭВМ.
а) выбор расположения контактных площадок; б) проведение электрических соединений; в) разводка печатной платы

много, что свидетельствует не столько о сложности прикладной системы, сколько о сложности самой области проектирования. Начинаящему пользователю системы можно рекомендовать следующую стратегию: начать с выполнения простых функций, игнорируя более изощренные приложения, а затем постепенно осваивать дополнительные возможности системы.

2.6. МОДЕЛИРОВАНИЕ ПРОЦЕССОВ

Инженеру, занимающемуся проектированием различных систем, приходится решать разнообразные задачи моделирования: радиоинженер заинтересован в средствах быстрого и эффективного моделирования поведения электронной схемы, инженер-механик—в расчете сил, приложенных к конструкции, специалист по теории массового обслуживания—в имитационном моделировании и т. п. С появлением ПЭВМ специализированное прикладное математическое обеспечение для моделирования процессов не только взрывообразно расширилось, но и приобрело новые качества, связанные с удобными вводом и графическим представлением данных. Общее число пакетов для конкретных инженерных задач расчета и моделирования в настоящее время составляет несколько сотен наименований.

Среди всего этого пестрого и разнообразного программно-обеспечения для решения задач моделирования на ПЭВМ можно выделить несколько наиболее распространенных типов программ. Расположим эти программы в порядке убывания распространенности:

- пакеты программ для логического моделирования работы цифровых электронных схем.

- пакеты программ для моделирования процессов в аналоговых электронных схемах.

- пакеты программ моделирования и синтеза систем автоматического управления.

- пакеты программ для моделирования механических свойств конструкций, тепловых режимов работы конструкций и анализа механики газов и жидкостей.

- пакеты программ для имитационного моделирования.

Рассмотрим кратко основные функции, реализованные в этих пакетах.

Логическое моделирование работы электронных схем. Пакеты для логического моделирования работы электронных схем обычно входят в программы для автоматизации проектирования. Такие пакеты включают в себя следующие основные процедуры:

- описание логических функций (алгоритма функционирования), выполняемых схемой.

задание выбора контролирующих тестов.
анализ результатов моделирования.
структурный синтез схемы и ее оптимизация.

Обычно анализируемые схемы представляют собой достаточно сложные устройства, имеющие узлы различного назначения: устройства управления, память, интерфейсные модули и т. п. При моделировании работы схемы решаются две основных задачи: собственно моделирование, заключающееся в подаче на вход устройства набора определенных событий и прослеживании реакции устройства на эти события, и синтез (оптимизация и создание оптимальной логической структуры, реализующей необходимые операции).

На этапе описания алгоритма функционирования схемы пользователь с помощью программ редактора принципиальной схемы создает ее изображение на экране и описывает основные функции, выполняемые модулями (сравнение, инверсия, дешифрирование и т. п.). После задания описания логических функций схемы пользователь может задать описание контролирующих тестовых воздействий на схему. Эти воздействия могут быть описаны в виде таблиц или в виде алгоритма их генерации. При первом методе на экране возникает таблица входов схемы и оператор может выбрать из библиотеки стандартных воздействий сигналы, подаваемые на эти входы (например, импульсы) и их последовательность во времени, т. е. в интерактивном режиме формировать подобие временной диаграммы воздействия. При задании алгоритма генерации входного воздействия пользователь должен с помощью специальных программ задать формализованный алгоритм генерации входных сигналов. На рис. 2.20 показан экран ПЭВМ при задании разработчиком набора входных сигналов для моделирования работы схемы. Слева на экране расположены текстовые окна, в которых разработчик задает имена сигналов, используемых в разрабатываемой схеме. Справа располагается меню, с помощью которого пользователь может выбрать ряд стандартных событий, например последовательность импульсов определенной частоты, и произвольным образом расположить их на временной диаграмме.

Процесс моделирования разбивается на элементарные такты. На каждом такте компьютер может выдать функцию логического состояния любой из точек анализируемой схемы, что обеспечивает цифровую имитацию прохождения логических сигналов в схеме. В результате моделирования пользователь может проверять последовательности сигналов в различных точках схемы, наблюдать набор временных диаграмм, т. е. моделировать на ПЭВМ процедуру подключения осциллоскопа или цифрового анализатора. В ряде пакетов для моделирования используется метод построения таблиц неисправностей. Например, в тех случаях, когда в результате моделирования было

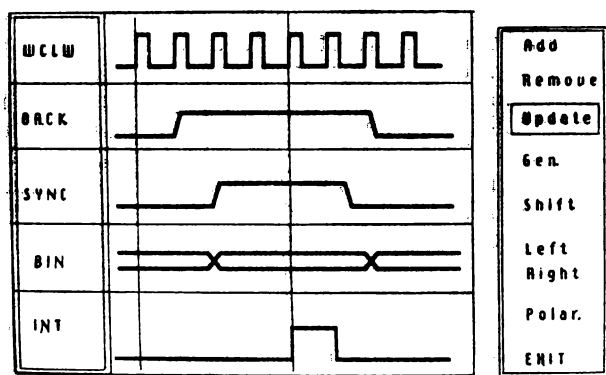


Рис. 2.20. Экран ПЭВМ при работе с системой логического моделирования электронных схем: редактор входных сигналов

получено отклонение от исходно заданной цели моделирования, или при обнаружении нарушений режима синхронизации существует возможность информировать разработчика о найденных неисправностях.

В пакеты для логического проектирования включены процедуры синтеза и оптимизации принципиальных схем. При решении задачи синтеза задается логическая функция, которую должно выполнять устройство, и при помощи специальных процедур синтеза, основанных преимущественно на булевой алгебре, производится реализация необходимой функции с помощью набора элементарных операций. При этом пользователь может сформировать «задание» системе, описав необходимые логические операции. Затем специальная программа выдает на экран принципиальную схему устройства, реализующего данную функцию. Это устройство может быть составлено из набора доступных модулей, выполняющих логические функции и имеющихся в базе данных системы, в частности на основе библиотеки стандартных микросхем. При этом одновременно может решаться и задача оптимизации выбора необходимых микросхем с точки зрения минимизации числа свободных выводов или потребляемой мощности и т. д. При решении задачи оптимизации уже имеющейся структуры с помощью процедур синтеза проверяется возможность использования более рациональных вариантов: уменьшение числа корпусов, использование других типов интегральных схем и т. п.

Логическое моделирование электронных схем—отдельная и большая тема. Несомненно, что удобные диалоговые системы проектирования, такие, как SmartPack фирмы Gould, Logic Instruments, MPESC (RCA), LogiCaps (Altera Corporation), FutureNet, RCad (Personal CAD systems) и др., в несколько раз повышают эффективность труда разработчика цифровых систем.

Моделирование процессов в аналоговых электронных схемах. Основной целью пакетов программ для моделирования аналоговых электронных схем является определение реакций цепи на заданные входные воздействия и расчет токов и напряжений в различных участках схемы. Вся специфика реализации таких систем на ПЭВМ заключается в максимальном использовании возможностей графики и диалога. Типичные пакеты программ для моделирования работы аналоговых электронных схем состоят из следующих основных компонентов:

- программы ввода данных о принципиальной схеме и редактирования (редактор принципиальной схемы).

- программы формирования уравнений состояния схемы,

- программы расчета частотных характеристик схем и временного анализа поведения схемы,

- программы для определения функций чувствительности схемы,

- программы синтеза электрических цепей и оптимизации параметров электронной схемы,

- программы работы с библиотекой элементов и схем (см. § 2.5).

Как выглядит типичная последовательность работы с такими системами?

Пользуясь пакетами программ для моделирования схем, пользователь вначале с помощью редактора принципиальной схемы задает ее вид. Этот процесс в интерактивных пакетах устроен достаточно изящно: передвигая курсор по экрану, можно выбирать из специального графического окна условные изображения элементов и двумя-тремя нажатиями клавиш соединять выбранные точки схемы (подробнее см. в § 2.5). Кроме того, пользователь может задать номиналы выбранных элементов или их типовые параметры из базы данных.

После задания принципиальной схемы в памяти ПЭВМ хранится информация об основных цепях, элементах и их топологических связях. Эта информация является исходной для программ формирования уравнений состояния схемы. Результатом работы этой подсистемы является формирование системы уравнений, описывающих заданную схему. Обычно этот этап работы скрыт от пользователя.

Теперь разработчик должен указать ряд параметров: тип используемых моделей элементов (линейные, линеаризованные, линейно-кусочные, табличные нелинейные), характер проводимого анализа (статический или динамический). Обычно в рамках систем моделирования работы электронных схем пользователь может, задав значения входных напряжений, двигаться курсором по различным точкам принципиальной схемы и определять значения токов и напряжений в выбранных узлах. В наиболее «эстетичных» системах при этом на экране возникает условное изображение тестера, который можно «включить».

чать» в различные участки схемы. Понятно, что в таком режиме у пользователя возникает ряд возможностей: модифицировав значения номиналов элементов или изменив их параметры, он может «пройтись» по характерным участкам схемы и определить токи и напряжения, как это делал бы инженер, собравший схему на макете.

Программы расчета частотных характеристик позволяют рассчитать амплитудно- и фазо-частотные характеристики цепи, основываясь на сформированных уравнениях состояния. Программы расчета временных характеристик рассчитывают вид переходной функции или импульсной реакции системы при заданных входных воздействиях (импульсе, единичном скачке, синусоидальном колебании, произвольной функции, заданной пользователем). При этом также существует возможность двигаться по различным точкам принципиальной схемы и наблюдать в них характерный вид переходного процесса: на экране возникает графическое окно, наложенное на изображение принципиальной схемы, и в нем отображается вид переходного процесса. Такая система обеспечивает значительную гибкость, так как в любой момент пользователь может менять условия функционирования схемы: задавать новые значения номиналов, различные входные воздействия и наблюдать за происходящими изменениями в реакции цепи.

Программы определения функций чувствительности схемы позволяют определить влияние параметров различных элементов схемы на характеристики ее работы. Обычно для определения чувствительности схемы вычисляются производные выходных параметров по значениям параметров элементов схемы, например исследуется влияние полосы пропускания схемы в зависимости от заданной емкости и т. п. При работе с этой программой пользователь в диалоговом режиме формирует задание, например выделяет на экране интересующую его емкость и параметр переходного процесса. В результате в графическом окне он может получить график зависимости данного параметра от емкости.

Программы синтеза электрических цепей и программы оптимизации параметров схемы обычно близки. При работе с этими программами пользователь может задать в графическом или аналитическом виде необходимую частотную характеристику или переходный процесс в цепи, начальные значения параметров компонентов, их допустимые изменения и технологический разброс, допустимые отклонения результирующей частотной характеристики от заданной. Теперь можно потребовать решить задачу синтеза и оптимизации: подобрать значения параметров компонентов, а иногда и модифицировать электронную схему, чтобы добиться заданной формы частотной характеристики или переходного процесса.

Таким образом, система моделирования аналоговых электронных схем в значительной степени избавляет разработчика от традиционных процедур расчета, пользования справочниками для определения параметров транзисторов, диодов, стабилитронов, утомительного (и не всегда возможного аналитически) расчета частотных характеристик, словом, позволяет не отвлекаться на рутинные, автоматические задачи, а в максимально короткое время находить новые решения.

Моделирование и синтез систем автоматического управления. Такие системы на ПЭВМ состоят из следующих основных блоков:

- программы построения моделей,
- программы проведения модельных экспериментов, оптимизации и синтеза системы автоматического управления,
- программы сопряжения с испытательным оборудованием,
- программы анализа результатов испытаний и идентификации систем.

Практически почти со всеми компонентами подобной системы мы уже встречались в параграфах, посвященных анализу экспериментальных данных, математическим расчетам и автоматизации. Рассмотрим более подробно основные компоненты системы моделирования процессов.

Программы построения моделей предназначены для диалогового задания модели разрабатываемой системы. При этом существует множество вариантов: пользователь может задать основные блоки системы и описать каждый из них передаточной функцией, ввести нелинейности, записать динамику входных и выходных параметров с помощью системы дифференциальных уравнений. Для того чтобы реализовать все это, в системах предусмотрена возможность использования библиотеки стандартных блоков автоматического регулирования, а также возможность диалоговой компоновки этих блоков в одну систему.

Как выглядит такой процесс? Пользователь может действовать двумя основными путями: проектированием «сверху вниз» и «снизу вверх». При первом подходе инженер компоует на экране структурную схему системы самого верхнего уровня. Предположим, что такая система состоит из трех блоков (рис. 2.21). Теперь каждый блок можно назвать своим именем (Блок 1, Блок 2, Блок 3) и задать последовательность прохождения сигнала в такой системе, соединяя созданные блоки при помощи специальной команды меню, расположенного в правой части экрана. Отметим, что у каждого блока могут быть несколько пронумерованных входов и выходов. Теперь зададим структурную схему блоков нижнего уровня. Например, на рис. 2.22 приведена структурная схема блока 1. Этот процесс можно развивать и дальше, все более усложняя схему установки. При проектировании «снизу вверх» последователь-

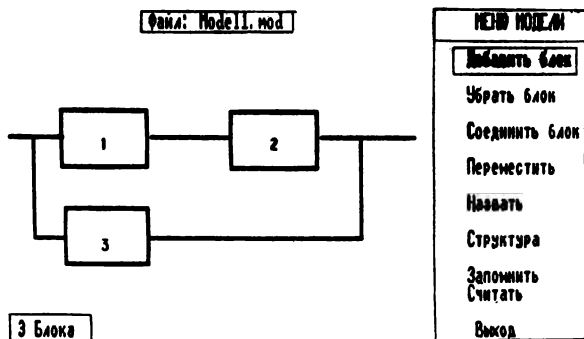


Рис. 2.21 Экран ПЭВМ при работе с системой моделирования систем автоматического управления: задание блоков модели

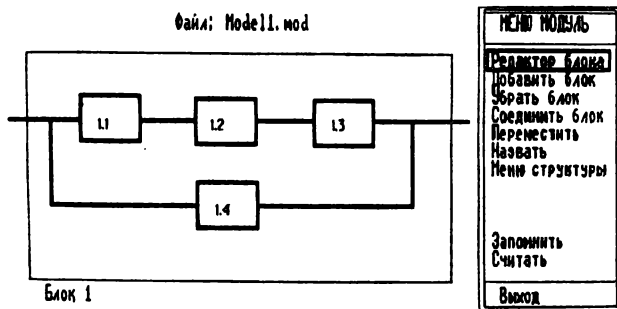


Рис. 2.22. Экран ПЭВМ в режиме редактирования модели блоков автоматического управления

ность действий обратна: вначале задаются узлы самого низшего порядка, которые затем объединяются в более сложные блоки и наращиваются. Пользователь не ограничен в выборе способа задания модели системы и может в любой момент комбинировать эти способы разработки: все блоки и модули запоминаются в специальной библиотеке.

На самом низшем уровне сложности расположены простейшие составные элементы системы: стандартные блоки системы автоматического управления. В типичной системе моделирования имеется библиотека стандартных блоков, состоящая из усилительных звеньев, дифференцирующих звеньев первого и второго порядка, инерционных элементов (интегрирующих звеньев), стандартных блоков, составленных из перечисленных выше, например звеньев, имеющих заданную частотную характеристику и описываемых алгебраическими уравнениями,

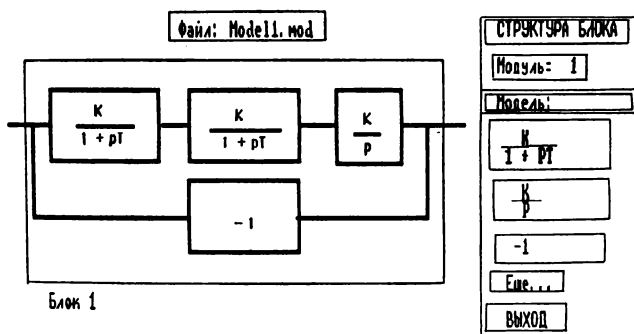


Рис. 2.23. Редактор элементарных блоков в системе моделирования

кусочно-линейными и типичными нелинейными функциями (экспоненциальными или тригонометрическими и др.).

Библиотека стандартных блоков в типичной системе моделирования соответствует их набору из вузовского учебника по автоматическому регулированию.

В библиотеку стандартных блоков нетрудно добавить любое звено. Типичные модули можно искать с помощью базы данных, комбинировать аналогично тому, как это было описано в § 2.5, посвященном системам автоматизированного проектирования. При задании структуры элементарных блоков модели (рис. 2.23) в правой части экрана возникает меню стандартных блоков, из которых легко выбрать необходимый и «поместить» его в соответствующий блок графического окна.

Модуль построения модели можно по аналогии с системами автоматизированной разработки чертежей назвать редактором модели. Таким образом, пользуясь системой построения модели, инженер с помощью «электронного конструктора» составляет на экране структурные схемы, пользуясь библиотекой «кубиков», объединяет эти модули, указывает взаимосвязи их входов и выходов.

Как только пользователем определена структура системы, заданы основные блоки, их взаимосвязи и передаточные функции, можно начинать этап моделирования. При моделировании возникает необходимость проверить, как откликается система на заданные входные сигналы, попытаться оптимизировать параметры системы или произвести синтез системы автоматического управления, например определить ее параметры, чтобы получить желаемый вид отклика системы. При этом пользователь может выполнять следующие основные операции:

рассчитывать отклики системы или любой ее части на заданные внешние воздействия,

оптимизировать заданные параметры системы,
проводить синтез системы автоматического управления.

Наиболее распространенной процедурой при моделировании является определение откликов системы на внешнее воздействие: пользователю необходимо знать вид переходного процесса в системе. Расчет откликов системы на заданные входные воздействия может производиться несколькими способами. В простейшем случае, когда система линейна, пользователь может выделить любую часть полной системы и потребовать рассчитать ее отклик на типовые внешние воздействия: импульсный сигнал, функцию единичного скачка, колебание определенной частоты. Существует также возможность задать произвольную входную функцию и определить реакцию системы. При расчете реакции системы указываются входы, выходы и тип входного воздействия.

В рамках приведенного примера для расчета переходной функции блока достаточно выбрать из специального меню вид входного воздействия и функцию построения переходной характеристики. При этом в графическом окне экрана появится кривая переходного процесса и список основных параметров (рис. 2.24).

Если в системе имеются нелинейные блоки, то предусмотрены процедуры линеаризации. Для систем, описываемых дифференциальными уравнениями, имеется возможность использовать набор алгоритмов интегрирования. Обычно в системы включают набор таких алгоритмов (методы Рунге—Кутты, предиктор—корректор и др.).

Другим типом задач, решаемых при моделировании, является оптимизация параметров модели. При этом задается

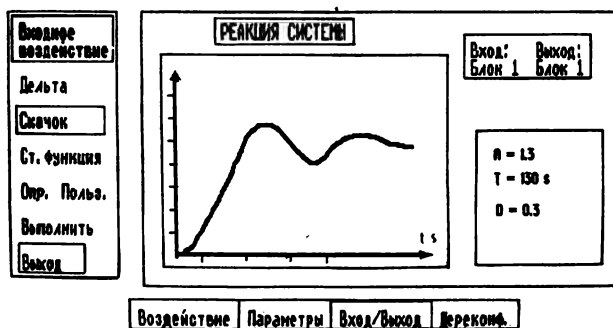


Рис. 2.24. Результат моделирования системы автоматического управления: расчет вида переходного процесса

оптимизируемый параметр (например, амплитуда выброса переходного процесса при подаче на вход единичного скачка) и требуется подобрать параметры основных блоков таким образом, чтобы минимизировать эту амплитуду. Для решения задачи оптимизации в пакеты моделирования процессов включают методы решения систем нелинейных уравнений и многомерной оптимизации (методы сопряженных градиентов, метод Ньютона, методы решения задач линейного программирования).

Для решения задачи синтеза систем автоматического управления САУ предусмотрены различные методы: построение годографов, диаграмм Найквиста, анализ передаточных функций, расчет оптимальных фильтров, разложение на собственные значения и т. п. Разнообразие методов здесь столь велико, что мы отсылаем читателя к современным руководствам по теории автоматического управления.

Таким образом, пользуясь программами моделирования, инженер может проигрывать различные варианты поведения модели: находить ее отклики на внешние сигналы, пытаться подобрать оптимальный режим работы или произвести синтез САУ.

Из систем моделирования на ПЭВМ наибольшую известность получили пакеты MATRIX/PC фирмы Integrated Systems и PC-Logger корпорации IBM. В литературе известно еще несколько десятков специализированных пакетов такого рода.

Моделирование механических свойств конструкций, механики жидкостей и газов. В последнее время на ПЭВМ большое распространение получают пакеты программ для расчета механических свойств конструкций и механики газов и жидкостей, предназначенные для использования инженерами-механиками. Разнообразие решаемых задач в этой области чрезвычайно велико: существует множество пакетов, ориентированных на анализ различных механических свойств объектов. В различных прикладных пакетах рассчитываются: несущая способность и прочность при статических нагрузках, динамических нагрузках, прочность валов и осей, резьбовых соединений, сварных соединений, сосудов, находящихся под давлением, и т. п. Если попытаться хоть как-то классифицировать структуру этих систем, то она будет выглядеть следующим образом:

- интерактивная надстройка для задания характеристик объекта исследования,

- система для задания типов внешнего воздействия,
- система для расчетов необходимых характеристик,
- программы представления данных,
- системы оптимизации параметров конструкции.

Работа с такими системами выглядит следующим образом. Инженер, пользуясь базой данных деталей и конструкций,

компонует модель исследования: выбирает стандартные детали (например, балки, соединительные пластины, диски, валы), задает их характеристики (материал, температура, упругие свойства, вязкость).

Задаются тип внешних воздействий: сосредоточенные или распределенные внешние силы, типы креплений, статические или динамические нагрузки, периодичность воздействия и т. д.

Рассчитываются необходимые характеристики: распределение статических или динамических механических напряжений, коэффициенты изгиба, смещения элементов конструкции, резонансные частоты, моменты инерции и центры тяжести, моменты сил, условия теплообмена, скорость и режим течения газа или жидкости, словом, практически все, что можно вычислить по формулам или рассчитать, решая системы дифференциальных уравнений. Многие пакеты представляют собой интерактивные надстройки над справочниками по расчетам деталей машин на прочность, объединенные с базой данных стандартных изделий.

Получаемые данные представляются в любой из форм, необходимых инженеру: графики, поверхности, таблицы.

В наиболее развитых пакетах начинают появляться экспертные надстройки для оптимизации параметров и выдачи рекомендаций, например анализируется выбранная конструкция и делается попытка перераспределения внешних сил, чтобы равномерно распределить поле напряжений или изменить режим работы системы для выбора оптимального теплового режима.

Поскольку разнообразие решаемых задач, как мы уже отмечали, чрезвычайно велико, приведем краткий обзор наиболее известных по публикациям пакетов для моделирования механических конструкций, работающих на персональном компьютере IBM PC XT. Любой из этих пакетов (или хотя бы все перечисленные) заслуживает, на наш взгляд, отдельной книги, так как разнообразие расчетных операций и сложность применяемых методов анализа являются предметом отдельного рассмотрения.

Пакеты STRESS/CM, STRESS FIELD/CM, C-BEAM фирмы GULF широко используются для анализа статических нагрузок конструкций. Пользуясь этими пакетами, инженер задает геометрию структуры, характеристики ее частей и приложенные нагрузки. Система рассчитывает поля напряжений и силы в двумерных и трехмерных структурах, при этом можно задавать дополнительные условия на смещения элементов конструкции, на силу тяжести. В пакете используется метод конечных элементов. Аналогичные возможности предусмотрены и в пакете STRESS and STRAIN фирмы DYNACOMP: пользователь может выбирать объекты из базы данных, задавать

нагрузки, кроме того, имеется возможность рассчитывать резонансные частоты конструкции.

Аналогичную структуру имеет пакет FATPACK фирмы Structural Software Systems. Пакет VBELT фирмы COADE позволяет рассчитывать параметры различных типов передач, шестерней, валов. При этом задается структура конструкции, например, крутящие моменты, количество дисков и их ориентация, углы зацепления зубчатых передач и т. п. Задаются и характеристики материалов: прочность, текучесть, предел выносливости при кручении. Пакет CEASAR II той же фирмы ориентирован на расчет трубопроводов: задаются геометрические характеристики труб, их материал, характеристики газа или жидкости, температура, вязкость и с помощью программ рассчитывается поле механических напряжений в стенках, условия теплообмена, характер течения жидкости. Специальная подсистема в этом пакете анализирует условия протекания жидкости в участках сочленения труб или в районах отверстий: разность давлений, изменение условий течения в зависимости от геометрических характеристик отверстия и т. п. Расчет практически всех мыслимых характеристик потоков жидкостей или газов обеспечивается с помощью пакета PESF фирмы Engineering Software: перепады давления, скорость и тип течения, потоки в многофазных системах, тепловые потери в стенках трубопроводов, условия перемешивания, обтекание потоком объектов сложной геометрии и т. п. К приведенному перечню можно добавить системы MicroShell (расчет напряжений в различных системах) и tMODE (анализ напряжений и тепловых потоков в многомерных системах методом конечных элементов). Для того чтобы читатель представил себе, как может выглядеть работа с подобными пакетами программ, на рис. 2.25 показан экран ПЭВМ

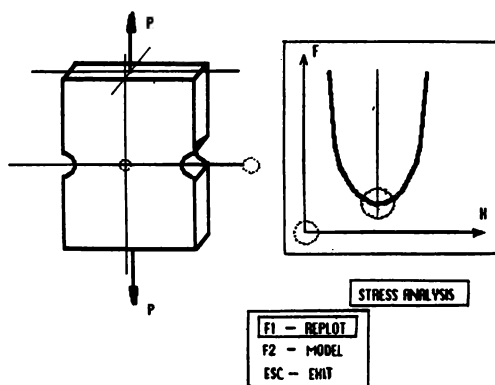


Рис. 2.25. Экран ПЭВМ при работе с системой моделирования механических свойств конструкций

в режиме расчета распределения напряжений в детали при ее растяжении.

Пакеты программ для имитационного моделирования. Имитационное моделирование широко используется при анализе работы сложных систем типа систем массового обслуживания, когда составить и решить аналитически уравнения, описывающие функционирование системы, не представляется возможным. При имитационном моделировании пользователь должен задать схему взаимодействия основных компонентов модели: какие объекты входят в общую систему, какой надежностью они обладают, сколько времени каждый из них требует на обслуживание той или иной задачи или операции и т. д. Пакеты для имитационного моделирования на ПЭВМ в настоящее время делятся на две основные группы: пакеты программ для оценивания надежности и времени обслуживания различных систем и языки имитационного моделирования.

В пакетах прикладных программ для оценивания характеристик сложных процессов пользователь с помощью своеобразного редактора модели формирует структурную схему исследуемой системы: рисует на экране условные изображения блоков и задает их характеристики (среднее время обслуживания, наработки на отказ, интенсивности поступающих потоков событий и т. д.). В результате работы модели оцениваются надежность системы, среднее время обслуживания заявок, его разброс, оптимальное распределение заявок на обслуживание для минимизации общего времени и др. Пользуясь таким пакетом, инженер может анализировать в диалоговом режиме функционирование системы и подбирать оптимальные варианты. Примерами таких пакетов прикладных программ являются системы PROJECT PLUS, RELIABILITY фирмы COADE, PSM, PIPS, QMS фирмы Mogeno Blocker Inc и др.

На рис. 2.26 показан экран ПЭВМ при работе с пакетом имитационного моделирования. Моделировалась система, имеющая три состояния: 0, 1 и поглощающее состояние 2 (попав в него, система уже не возвращается обратно). Такой модели может соответствовать какой-либо технологический процесс, например отказы одновременно двух резервирующих систем, которые выводят из строя всю систему. Заданы интенсивности переходов, в графическом окне получены результаты моделирования: оценивалась вероятность попадания системы в поглощающее состояние с течением времени. Как видно из рисунка, моделирование показывает, что через три временные единицы эта вероятность стабилизируется на уровне около 0,9.

Из языков имитационного моделирования для ПЭВМ наибольшее распространение получил GPSS (версия фирмы MinuteMan Software), являющийся стандартом для ЭВМ се-

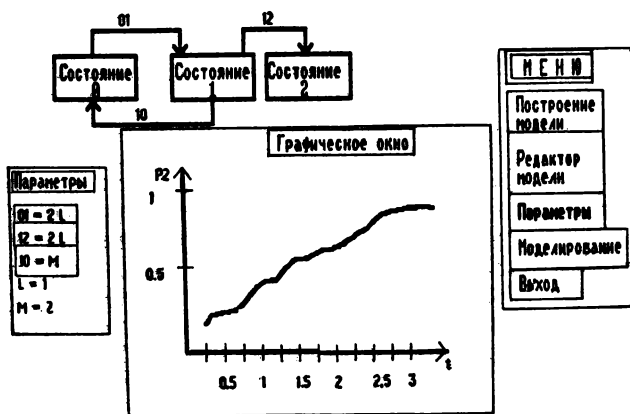


Рис. 2.26. Экран ПЭВМ при работе с пакетом программ имитационного моделирования

рии ЕС. С помощью этого языка можно решать различные задачи имитационного моделирования: в нем содержится набор операторов, описывающих последовательность обработки событий, комбинируя которые, можно организовывать имитацию функционирования практически любой сложной системы.

Поскольку имитационное моделирование является довольно специфической областью и в литературе имеется множество книг и руководств по языку GPSS, то на этом закончим наш краткий обзор прикладных систем моделирования для ПЭВМ.

Читатель, конечно, должен понимать, что приведенный здесь перечень пакетов отнюдь не отражает всего разнообразия подобных систем: каждый год издаются каталоги программного обеспечения, содержащие десятки новых наименований. Мы надеемся, что приведенная в этом разделе информация поможет читателю сформировать общее представление о подобных пакетах и, быть может, вдохновит его на создание специализированных систем такого рода в своей области.

2.7. БАЗЫ ДАННЫХ

Во многих инженерных и организационных задачах возникает ситуация, при которой в ПЭВМ необходимо держать большой объем информации. Простейший, хотя, по-видимому, наиболее типичный для большинства лабораторий пример — ведение картотеки радиодеталей: наличие определенных номиналов резисторов, типов микросхем и т. п. Средний объем такой библиотеки обычно не менее нескольких сотен наименований и поиск сведений о нужных деталях часто превращается в мучительную процедуру.

Мы уже упоминали про базы данных (БД), являющиеся составной частью различных прикладных систем проектирования и моделирования. Читатель, который прочел эти параграфы, по-видимому, понял, что без специального математического обеспечения, позволяющего отыскивать в больших массивах данных нужную информацию, большинство прикладных систем потеряли бы до 80% своей мощности. Но БД, непосредственно входящие в проблемно-ориентированные пакеты, обычно являются довольно специфическими системами, «настроенными» на конкретные применения. Чтобы пользователь мог решать свои задачи работы с БД широкого профиля, для ПЭВМ разработано разнообразное программное обеспечение.

Вначале рассмотрим общие понятия, необходимые для дальнейшего изложения. Каким образом задать набор сведений об объекте, описывающий все его свойства, которые могут когда-нибудь пригодиться в решаемой задаче? Так, при ведении каталога интегральных схем инженера могут интересовать наименование микросхемы, число выводов, напряжение питания, тепловыделение, масса, количество и дата поступления. Если просто записать все эти данные в файл, например с помощью текстового редактора, то, конечно, можно с помощью команды поиска найти все сочетания символов типа K1804, но не более. Для того чтобы компьютер «понял» вводимые данные, необходимо разработать структуру БД. Каким наиболее логичным образом можно было бы описать информацию об имеющихся в наличии микросхемах?

Информация о конкретных параметрах, записанная в файл, называется записью. Обозначение «Напряжение питания» является полем. Совокупность полей, описывающих объект, и информации о типах данных в этих полях образует структуру базы данных. Таким образом, чтобы компьютер понял, о какой информации идет речь и как она организована, мы должны задать структуру записи: указать наименования полей и тип записываемой в них информации. Для нашей задачи полями являются:

Наименование микросхемы

Число выводов

Напряжение питания

Тепловыделение

Количество

Масса

Дата поступления

Всего мы задали 7 полей, причем первое из них описывается сочетанием символов, например K1804, а последующие—числами. Поэтому надо каким-то образом указать компьютеру, что в первом поле записываются символьные переменные, во втором—

целые числа и т. д. Кроме того, мы должны задать допустимые размеры записи и количество цифр, например наименование микросхемы не должно превышать сочетания 10 букв и цифр, напряжение питания—трех цифр. Рассматриваемую процедуру называют созданием структуры записи в БД. Уже введенная запись может выглядеть следующим образом:

Наименование микросхемы	K1804BY1
Число выводов	28
Напряжение питания, В	5
Тепловыделение, мВт	600
Количество	24
Масса, г	10
Дата поступления	11/10/87

Теперь, когда задана структура хранящейся информации, можно решать самые разнообразные задачи по извлечению информации. Предположим, что в файле MICRO. DBF содержится информация о 100 типах микросхем, имеющихся в наличии. Теперь можно запросить информацию о всех микросхемах, полученных лабораторией до 1/01/87. Условие поиска можно усложнить, например объединить только что заданное условие с тем, что микросхемы должны иметь напряжение питания 5 В и т. п. Возможности выбора различных данных из БД ограничены только ее размерами и фантазией пользователя.

Наиболее типичными применениями БД в инженерных приложениях являются уже упомянутые каталоги деталей, конструкций, их свойств, хранение описаний проводимых экспериментов, банки данных результатов исследований и моделирования. Кроме того, существует огромное поле приложений БД в экономических и организационных работах: хранение информации о финансовых документах и их обороте, данных о сотрудниках, транспортных потоках и др.

Типичная структура программного обеспечения для работы с БД на ПЭВМ состоит из трех основных компонентов:

диалоговой надстройки, позволяющей пользователю задать структуру БД, проводить ввод, поиск и выдачу интересующей его информации:

командного языка для работы с БД, позволяющего писать программы для выполнения специфических операций;

интерфейса языка управления БД, для того чтобы пользователь мог включать функции работы с БД в свои собственные программы.

Функции диалоговой надстройки и языка управления БД обычно перекрываются. Рассмотрим вначале основные функции при работе с БД, а затем опишем их реализацию на языке управления данными.

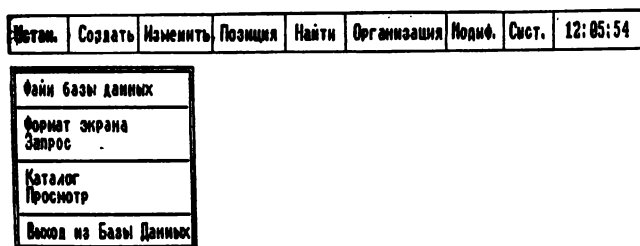


Рис. 2.27. Меню системы управления базой данных ПЭВМ

При загрузке программы для работы с БД на экране появляется меню (рис. 2.27), состоящее из нескольких разделов. Вызывать различные разделы можно с помощью клавиш управления перемещением курсора. Обычно эти разделы имеют следующее назначение:

задание параметров—выбор файла данных, формата вывода данных на экран и т. п.;

создание новой структуры файла данных, характера ее представления на экране, формы выдачи отчета;

редактирование записей в БД—изменение, стирание, добавление, просмотр и т. п.;

поиск данных, хранящихся в БД и удовлетворяющих заданным условиям;

извлечение необходимой информации из БД;

организация файла БД;

математические расчеты над данными, хранящимися в БД;

системные функции.

Рассмотрим основные функции, реализованные в перечисленных разделах.

В разделе «Задание параметров работы» пользователю предоставляется возможность выбрать из имеющихся стандартных модулей формат экрана, тип информации, которую он будет искать в файле, набор файлов данных, с которыми он будет работать. Если пользователь хочет сам определить все эти параметры, то он должен обратиться к разделу «Создание новой структуры».

Предположим, что Вы решили создать новый файл данных и выбрали режим «создание новой структуры файла данных». При этом на экране появляется табличка, в которой имеются следующие параметры:

Имя поля Тип поля Размер поля

Теперь можно начать ввод информации о структуре. Для этого в каждой колонке таблицы записываем необходимую

информацию. Для нашего примера она может выглядеть следующим образом:

Имя поля	Тип поля	Размер поля
Наименование микросхемы	Character	10
Число выводов	Integer	3
Напряжение питания	Real	4
Тепловыделение	Integer	6
Количество	Integer	5
Масса	Integer	5
Дата поступления	Date	

Мы указали, что наименование микросхемы является символьной переменной и не должно занимать более чем 10 позиций текста, число выводов, тепловыделение, количество и массу мы решили отображать целыми числами, а напряжение питания—действительными числами (например, 7, 5 В), для времени поступления детали выбран специальный тип переменной «Дата», который указывает на то, что в этом поле будет записана информация о дате в формате месяц/число/год.

После того как ввод информации о структуре данных закончен, пользователь может перейти к процедуре ввода записей. При этом на экране возникает табличка, в левой части которой выделены названия полей, а в правой цветом или яркостью отмечены участки, в которых пользователь может вводить данные (рис. 2.28). Например, для поля «Число выводов» справа будет выделено три позиции, для поля «Тепловыделение»—пять и т. д. В системах имеются широкие возможности для выбора формата вводимых данных: пользователь может выбрать способ расположения вводимых полей на экране, изменить их цвет, нарисовать специальные квадратики, в которых будет вводиться информация, и т. п., т. е. выбираться оптимальный экранный интерфейс для ввода и представления информации.

Устан.	Создать	Изменить	Позиция	Найти	Организация	Модиф.	Сист.	12:05
--------	---------	----------	---------	-------	-------------	--------	-------	-------

Наименование	K1804BY1
Число выв.	24
Напряжение	5
Тепловыд.	600
Количество	24
Вес	10
Дата пост.	11/10/87

Рис. 2.28. Пример ввода информации в базу данных на ПЭВМ

В режиме «Создание новой структуры» можно также определить и ряд дополнительных форматов, например вид выдачи отсчета исходя из имеющихся данных. В чем заключается смысл такой операции? Число различных полей в записи может быть довольно большим, но в зависимости от ситуации пользователя системы может интересовать только их малая часть. Например, в одном случае инженеру может быть необходимо найти только наименования микросхем, поступивших в лабораторию в заданный период, а в другом — определить только количество микросхем определенного типа. Для того чтобы не загромождать экран дисплея или распечатку ненужными данными, можно заранее указать системе, что именно будет интересовать пользователя в том или ином случае и каким образом необходимо разместить выдаваемую информацию.

На рис. 2.29 приведен пример вывода информации о микросхемах, при котором из информации, хранящейся в БД, оставлены лишь пункты, относящиеся к наименованию, напряжению питания, количеству и дате поступления. Для этого предусмотрена специальная процедура «Создание формата представления данных» и меню «Создание формы отчета». В этом меню пользователь задает: поля, которые ему необходимо вывести, их расположение на распечатке или на дисплее и в некоторых случаях математические операции, задаваемые над этими полями. Например, в отчете может быть необходимо не только перечислить все микросхемы, но и подсчитать их общее число. В задаваемый отчет можно включить и логические условия, по которым необходимо получать выборку данных. Например, частой операцией при работе с БД деталей может быть получение информации об общем числе микросхем заданного типа. Тогда этот тип можно ввести в качестве условия выборки в форму отчета. Созданную форму отчета можно запомнить в специальном файле, а при повторном вызове программы работы с БД останется только указать имя этого файла и полу-

Устан.	Создать	Изменить	Позиция	Найти	Организация	Модиф. Сист.	12:05
--------	---------	----------	---------	-------	-------------	--------------	-------

Наименование
K1004031

Напряжение питания
5 В

Количество
24

Дата поступления
11/10/87

Рис. 2.29. Экран ПЭВМ при выводе данных из базы данных в заданном формате

чить необходимую информацию, удобно расположенную на листе бумаги и содержащую ответы на все поставленные вопросы. Кроме того, в дополнительном файле можно запомнить набор часто повторяющихся запросов к БД.

В режиме «Редактирование записей» пользователь может изменять уже имеющийся файл данных. Возможности, предоставляемые системой редактирования, близки к возможностям обычного текстового редактора: можно передвигаться курсором по экрану, стирать любые записи или добавлять новые, редактировать уже имеющуюся запись, менять расположение выделенных участков текста. Например, для того чтобы изменить имеющееся в наличии число микросхем, необходимо выполнить следующую последовательность операций:

- войти в режим редактирования записи,
- клавишами управления курсором найти необходимую запись,

- двигаясь курсором, найти поле «Количество»,
- записать новое число микросхем,
- дать команду на запись внесенных изменений.

В режиме редактирования существует ряд дополнительных возможностей: просматривать и редактировать одновременно одну или несколько записей, дать команду на замену выбранного значения сразу во всех записях, потребовать изменить одно сочетание символов и цифр на другое.

В разделе «Поиск данных» пользователь может найти запись, удовлетворяющую заданному условию. Для этого необходимо выполнить примерно такую последовательность операций: выбрать режим поиска, при этом на экране появится окно, в котором будет приведен набор полей структуры. Предположим, нас интересуют все микросхемы, имеющие массу 10 г. Выберем из этого окна поле «Масса». Теперь на экране появится другое окно, в котором приведен набор условий поиска (равен, больше чем, меньше чем и др.). Выберем условие «равен». В окне появляется приглашение к вводу числового значения (рис. 2.30). Запишем туда число 10 и дадим возврат каретки. На экране появится первая запись, удовлетворяющая этому условию. Нажав на клавишу перевода страницы, увидим следующую запись и т. д. В этом разделе можно также поставить БД на запись с заданным номером и пропустить заданное число записей, удовлетворяющих введенному условию поиска.

В разделе «Извлечение информации» пользователь может «вынуть» из БД любой набор записей, удовлетворяющих введенным условиям поиска, и представить их в заранее подготовленном виде. Условия на поиск данных вводятся аналогично тому, как это описано в предыдущем разделе. Отметим только, что эти условия могут быть достаточно сложными и включать набор математических функций и логических операций над различными полями. Для этого последовательно ука-

Устан.	Создать	Изменить	Позиция	Найти	Организация	Модиф.	Слот.	12:05:54
--------	---------	----------	---------	-------	-------------	--------	-------	----------

НАИМЕНОВАНИЕ
ЧИСЛО ВНЕ.
НАПРАВЛЕНИЕ
УСЛОВИЕ
КОЛИЧЕСТВО
ВЕС
ДАТА ПОСТ.

ИМЯ ПОЛЯ
СХЕМЫ -> ВЕС

Список
Показать
Отчет
Метка

Сумма
Среднее
Подсчет

= Равен 10
{ : Меньше или равен
{ : Меньше чем
} : Больше чем
} : Больше или равен
() Не равен

Выполнить команду
Выбор диапазона
Список полей
Условие поиска
Диапазон поиска

Рис. 2.30. Работа с системой управления базой данных в режиме поиска данных, удовлетворяющих выбранному условию

зываются условия, накладываемые на поля, и их логические связи. Например, условие означает «Найти все микросхемы «Масса» = 10 . AND. «Дата» > = 01/01/87 с массой 10 г, полученные после 01.01.1987». После задания условий и выполнения операций извлечения данных получается список записей, который можно вывести на экран, печатающее устройство или записать на диск в выбранном формате.

Раздел «Организация файла» позволяет сделать три основных операции: сортировку, индексирование и слияние файлов. Сортировка заключается в том, что пользователь выделяет интересующее его поле и располагает записи, основываясь на значении поля в алфавитном порядке (если поле символьное), дате или по возрастанию или убыванию чисел (если поле числовое). Индексирование позволяет выделить одно из полей в качестве индексного и затем по умолчанию проводить операции поиска и сортировки исходя из значений данного поля. Слияние файлов позволяет использовать данные, содержащиеся в нескольких файлах БД. Для выполнения этой процедуры необходимо указать набор исходных файлов, те поля в каждом из них, которые необходимо сохранить, количество записей и, возможно, условия на эти записи и новый файл, в котором будет храниться информация. Эта возможность очень важна для ряда приложений, в которых необходимо, пользуясь существующими данными, получать их новые сочетания. Например, предположим, что в БД имеется файл, содержащий поле «Наименование микросхемы» и поле «Стоимость», отсутствующее в исходной структуре. Тогда, воспользовавшись процедурой слияния файлов, мы можем подготовить новый файл, содержащий сведения о стоимости, массе и наименовании микросхемы.

В меню «Математические расчеты» пользователю предоставляется возможность выбора числовых значений полей, условий и математических функций. Например, базе данных можно дать следующее задание:

выделить поля «Тепловыделение», «Количество»

выделить условие на поле «Напряжение питания» ≤ 12 В.

потребовать рассчитать сумму произведений тепловыделения на количество микросхем для всех схем, удовлетворяющих данному условию.

В наиболее развитых системах управления базами данных имеется большое число математических функций, позволяющих вычислить практически любое математическое выражение.

В разделе «Системные функции» пользователь обычно может, не выходя из системы управления БД, выполнять некоторое подмножество команд ОС: просматривать каталог файлов, копировать, стирать и переименовывать файлы и т. п.

Диалог с описанной системой обычно достаточно прост и приятен, в нем максимально используется естественное расположение данных в виде таблиц в различных участках экрана, реализован режим встроенных подсказок. Однако при частом выполнении одних и тех же операций, особенно если для извлечения информации из БД требуется большое число этих операций, диалоговая надстройка теряет свою привлекательность. Кроме того, во многих применениях возникает необходимость выполнять различные операции с БД циклически, проверять условия на прекращение цикла и т. п. Эти обстоятельства привели к тому, что в большинстве СУБД на ПЭВМ существует специальный командный язык манипулирования данными. С помощью этого языка можно написать достаточно сложные программы. Обычно язык манипулирования данными на ПЭВМ существует в двух вариантах: непосредственно встроенный в БД (интерпретирующий режим) и как независимое средство разработки (с использованием специальных компиляторов и программ сборки). В языке манипулирования данными реализованы все рассмотренные нами функции диалоговой надстройки. Приведем несколько примеров. Для того чтобы указать системе файл, с которым она будет работать, достаточно дать команду

USE MICRO.DBF

Для того чтобы указать системе, какой номер записи ей необходимо обрабатывать в данный момент, предусмотрена команда

GO RECORD n

Для того чтобы найти в БД запись, удовлетворяющую условию, достаточно дать команду FIND, например

FIND Масса = 10

В языке манипулирования данными предусмотрены циклы и операторы условного перехода. Например, следующий фраг-

мент программы будет проверять условие на напряжение питания микросхемы и, если оно выполняется, то тепловыделение микросхемы будет суммироваться в специальной переменной ТЕПЛО, в противном случае программа будет переходить к следующей записи:

```
USE MICRO.DBF /* Используем файл MICRO */
ТЕПЛО = 0
ПОЛЕ = 0 /* Инициализируем переменные ТЕПЛО,
ПОЛЕ */
GO RECORD 1
ЗАПИСЬ = 1
DO WHILE.NOT.EOF /* Повторять пока не конец файла */
  IF Напряжение <= 7 THEN
    STORE Тепловыделение TO ПОЛЕ
    ТЕПЛО = ТЕПЛО + ПОЛЕ
    ЗАПИСЬ = ЗАПИСЬ + 1
  ELSE
    ЗАПИСЬ = ЗАПИСЬ + 1
  ENDIF
  GO RECORD ЗАПИСЬ
ENDDO
WRITE ЗАПИСЬ /* Пишем значение на экран */
END
```

Читатель, знакомый с программированием, видит, что создание программы на языке манипулирования данными практически ничем не отличается от обычного программирования. Теперь созданную программу можно обозначить каким-нибудь именем и при необходимости вызвать простым указанием этого имени. С помощью языка манипулирования данными можно составить самые разнообразные программы, ориентированные на конкретные приложения. К сожалению, стандарта на такой язык пока не существует и в каждой системе используется свой диалект и свои правила манипулирования данными.

Но и сам язык манипулирования данными все же не дает многим пользователям необходимой свободы. Дело в том, что в ряде случаев необходимо встраивать функции работы с БД в прикладные программы, написанные на языках высокого уровня. С этой целью ряд фирм выпускает специальные библиотеки программ, содержащие ссылки на вызовы основных примитивов языка манипулирования данными из языков программирования.

На ПЭВМ, совместных с моделью IBM PC, разработано большое число программ для работы с БД. Наибольшую популярность среди них имеет пакет dBASE (в настоящее время распространена модификация dBASE III Plus) фирмы Ashton Tate. Среди других пакетов известны системы Condor 3, RBASE 4000, Paradox, Oracul, pfs-File, Perfect Filer, KnowledgeMan. Последний пакет резко выделяется из остальных тем, что в нем

БД соединена с программами обработки электронных таблиц, графикой, множеством математических функций (например, итеративным решением систем уравнений, пакетом статистической обработки и др.). В качестве примера программ для обеспечения интерфейса между БД и языками высокого программирования можно назвать систему dBaseC для связи системы dBASE с языком Си.

2.8. БАЗЫ ЗНАНИЙ И ЭКСПЕРТНЫЕ СИСТЕМЫ

Программное обеспечение, о котором пойдет речь в этом параграфе, на сегодняшний день является наиболее модным: 80-е годы охарактеризовались бумом в области экспертных систем и искусственного интеллекта. В то же время это направление стоит явно на одном из последних мест в области реальных практических результатов: надежды на то, что экспертные системы в скором времени заменят человека в выработке сложных решений явно не оправдываются, а многие программы, демонстрирующие «интеллект», вызывают сильное желание нелестно высказаться по поводу тупости компьютеров. Так, одним из первых знакомств авторов с экспертной системой на ПЭВМ был диалог с надстройкой, содержащей несколько сотен правил для выдачи рекомендаций специалистам по выбору места работы. Система задавала нам вопросы о характере наших предпочтений при выборе различных курсов в области математики, физики, жизненных интересов (типа желания найти место работы с максимальной зарплатой или наиболее соответствующее научным интересам). Наконец, в качестве рекомендации авторам был выдан совет заняться... финансовыми расчетами.

Несдержанная реклама последних достижений в области искусственного интеллекта создала ложные ожидания. Все встает на свои места, если относиться к этой области как просто к полезной составной части программного обеспечения, ускоряющей и упрощающей работу с набором формализованных правил точно так же, как БД ускоряют и упрощают работу с большими массивами информации.

Еще раз хотим подчеркнуть: ЭВМ обгоняет человека только в точности, объеме и скорости выполняемых математических операций. Операции, которые выполняет экспертная система, относятся к автоматизации логических выводов. По-видимому, каждый читатель неоднократно запутывался, когда ему приходилось делать даже несложные логические выводы. Например, рассмотрим простой житейский пример, связанный с неисправностью телевизора.

Правило 1. Если нет изображения и дело происходит днем, то сломался видеоблок, в остальных случаях будем считать, что видеоблок исправен, т. е. изображение не показывают.

Правило 2. Если нет звука, то или сломалась звуковая часть или в данный момент по телевидению показывают заставку.

Правило 3. Если часы показывают ночь и вы находитесь в Москве, то сейчас ночь, а если вы во Владивостоке, то сейчас день, и наоборот.

Очень просто, не правда ли? А теперь попробуйте быстро ответить на два простых вопроса.

Что случилось с телевизором, если часы показывают ночь, звук есть, изображения нет. Вы находитесь во Владивостоке?

Что случилось с телевизором, если Вы во Владивостоке, нет звука, нет изображения, часы показывают день?

По-видимому, Вам все же пришлось потратить несколько секунд на логический разбор этих ситуаций, хотя решение однозначно и четко задано системой формальных правил: в первом случае у телевизора явно сломан видеоблок, а во втором — звуковой блок. Теперь достаточно представить прослеживание цепочки хотя бы из двух десятков правил, чтобы понять трудность такой задачи.

В настоящее время известно несколько принципов построения экспертных систем: системы, основанные на правилах, и системы, использующие глубинные представления. Первые для своей работы требуют набора формальных логических правил типа приведенных нами, вторые, использующие глубинные представления, основаны на представлении информации в виде так называемых «фреймов» и значительно более сложны, чем системы, основанные на правилах. С помощью глубинных представлений можно моделировать подобию абстракций, пытаться находить аналогии между процессами, устанавливать их временные взаимосвязи в последовательности событий и т. п. Однако эти системы пока еще только начинают выходить за стены лабораторий. Распространение получили наиболее простые и прагматические экспертные системы, основанные на правилах. С помощью таких систем можно делать на основе набора правил логические выводы, причем если системе не будет хватать информации для вывода, она будет запрашивать ее у пользователя.

В инженерных и технических приложениях, задачах медицинской диагностики, разведке полезных ископаемых экспертные системы, основанные на правилах, зарекомендовали себя очень хорошо. Это связано с тем, что для принятия решений в этих областях часто требуются большие, но достаточно четкие наборы формальных правил, с которыми компьютер справляется лучше человека.

Наибольшая сфера применения экспертных систем в технических областях — решение задач технической диагностики и управления приборными комплексами и технологическими процессами. Это и понятно: в обоих случаях существует достаточно четкая формализация необходимых действий.

Например, в инструкции по диагностике ПЭВМ записаны следующие правила:

если в точках разъема 8 и 10 отсутствует тактовая частота 4,77 МГц, то проверить точку 1;

если в точке 1 отсутствует частота 14,318 МГц то неисправна микросхема генератора частоты;

если при включении питания в точке 3 не происходит перехода сигнала из низкого уровня в высокий и снова в низкий, неправильно установлена перемычка 1

и т. д.

Для анализа сложной схемы может быть необходимо несколько сотен правил. Понятно, что экспертные системы такого рода можно было бы объединить с системой проектирования. Например, инженер с помощью пакета моделирования рассчитал значения резонансных частот колебаний механической системы и хочет их уменьшить. В экспертной системе может быть записан примерно следующий набор правил:

если можно увеличить массу конструкции, то резонансная частота уменьшится в соответствии с формулой номер 44;

в противном случае необходимо проанализировать возможность изменения формы конструкции или материала изделия;

если резонансные колебания возникают в результате течения жидкости по трубопроводам, то перейти к правилам 12 и 13 для понижения частот колебаний.

Если данные из БД проекта можно каким-то образом передать в экспертную надстройку, хорошо продумав сопряжение этих систем, то инженер получает в свое распоряжение электронный справочник с записанными рецептами преодоления тех или иных технических проблем. При этом существенно, что если исходных данных недостаточно для ответа на поставленный вопрос, система сама определит недостающую информацию и запросит ее у пользователя.

Каковы основные принципы работы экспертной системы? Предположим, что у нас имеется набор правил для диагностирования некоторой электронной схемы, например напряжения в заданных точках схемы в различных ситуациях. Пользуясь этими правилами, можно проводить различные процедуры логического вывода. На этом шаге существует значительное разнообразие возможностей. Рассмотрим вначале простейшую ситуацию: в системе имеются значения всех напряжений, необходимых для диагностирования схемы. В этом случае программа вывода найдет соответствующие записи и просто «пропустит» их через введенный набор правил. Если хотя бы одно из правил окажется выполненным, то решение задачи найдено. При этом вполне может оказаться, что часть данных о напряжениях в узлах схемы избыточна.

При проведении процедуры вывода в подавляющем большинстве случаев известна не вся информация. Поэтому для того, чтобы оптимизировать число запросов к БД или к пользователю, необходима оптимизация логических выводов. Наиболее распространена система обратного логического вывода. В чем заключается смысл такой процедуры? Предположим, что нам известна конечная цель (например, определение неисправной микросхемы). Тогда мы можем выбрать из базы знаний правило, определяющее факт неисправности любой микросхемы. Если системе неизвестны значения необходимых напряжений в заданных точках, то найдем правила, определяющие напряжения в этих точках. Если такие правила найдены, но в них появляются новые неизвестные параметры, то найдем все правила, в которых определяются их значения, и так до тех пор, пока не переберем весь набор правил. Теперь запросим значения переменных, необходимых для оценивания самого последнего правила в нашей цепи, и если это правило выполняется, то проведем всю цепочку выводов в обратном порядке.

Противоположным является прямой логический вывод. В этом случае проверка выполнения правил начинается с тех переменных, которые известны системе. Если конечная цель вывода неизвестна, то такой поиск легко может завести нас в тупик. Так, пользуясь прямым выводом, мы можем найти напряжения в наборе узлов схемы, но не определим факта неисправности. Прямой вывод, однако полезен с той точки зрения, что он позволяет определить все переменные, которые могут быть получены исходя из начальных.

Дополнительная возможность при реализации логического вывода состоит в том, что мы можем присваивать каждому из правил определенную степень доверительности, тем самым вводя неопределенность в решение. Таким образом, в задачу вводится возможность существования альтернатив. Тогда система логического вывода должна определить несколько наиболее вероятных решений и выдать их значения с заданной вероятностью. Например, для рассматриваемой задачи результат работы системы может выглядеть следующим образом: с вероятностью 0,7 повреждена микросхема 3, с вероятностью 0,5 повреждена микросхема 5, с вероятностью 0,3 оборвана цепь питания.

Принцип вычисления вероятностей обычно крайне прост: вероятности заданной цепочки правил перемножаются.

Таким образом, с помощью системы логического вывода можно приходиться к заключениям о состоянии исследуемого объекта. В большинстве экспертных систем предусмотрена возможность объяснения пользователю, почему система пришла к тем или иным выводам. Так, на вопрос, почему повреждена микросхема 3, можно получить ответ примерно следующего вида:

«Так как напряжение в точке 1 меньше 5 В (правило 3), то напряжение в точке 2 также меньше 5 В (правило 4), и так как напряжение в точке 9 равно нулю, то из правила 1 следует, что микросхема 3 неисправна»

Описанная процедура работы экспертной системы может быть реализована в нескольких вариантах. Простейшим из них является полностью диалоговый принцип построения, при котором ПЭВМ выступает в виде своеобразного советчика, запрашивая у пользователя информацию в текстовой форме. Для реализации прикладных систем большей эффективностью обладает метод при котором экспертная надстройка может быть связана с БД и с прикладной программой поиска. При этом информация, необходимая для принятия решения, ищется в БД, а вопросы пользователю задаются только в тех случаях, когда исходной информации в БД недостаточно.

Что необходимо сделать инженеру для того, чтобы создать автоматизированную систему принятия решений? Эта задача довольно сложна: необходимо выделить основные параметры решаемой задачи, как-то сообщить их экспертной системе, задать набор правил над заданными переменными и, наконец, произвести процедуру логических выводов. Для решения этой группы задач на ПЭВМ разработан ряд экспертных систем.

Какова типичная структура такой экспертной системы на ПЭВМ и что можно сделать с ее помощью? Хорошая экспертная система состоит из следующих интегрированных между собой компонентов:

- системы управления базой знаний,
- базы данных,
- программы логических выводов, запросов и объяснений,
- языка программирования,
- интерфейса к процедурам пользователя.

С помощью такой экспертной системы можно выполнить следующие операции:

- задать числовые, текстовые и логические переменные и систему правил вида ЕСЛИ = ТО, т. е. определить базу знаний;
- поддерживать базу данных, имеющих отношение к решаемой проблеме;

- объяснить системе, что параметры, необходимые для базы знаний, можно запрашивать у пользователя в диалоговом режиме или искать в БД;

- проводить логические выводы, при необходимости объясняя их результат;

- программировать последовательность работы с системой, используя встроенный язык;

- использовать результаты проведенных решений как параметры для выполнения специфических программ пользователя.

Прежде всего, при работе с экспертной системой пользователь должен определить набор переменных, необходимых для задания набора правил. Эти переменные могут быть определены в памяти или заданы с помощью ссылок на БД.

Принципы работы с БД мы уже рассматривали. Все они сохраняются и в рамках экспертной системы. Пользователь может выделить поля записей в БД и присвоить им имена условных переменных или задать новые переменные. Например, для диагностирования повреждений в электронной схеме можно задать напряжения в узлах U1, U2 и т. д. Эти напряжения могут быть первоначально записаны в БД совместно с другими параметрами схемы. В этом случае мы с помощью специальной процедуры укажем, что переменным U соответствует значение поля «Напряжение» определенного файла БД. На рис. 2.31 показан экран ПЭВМ при работе с такой системой.

Основными компонентами программного обеспечения экспертной системы, отличающим ее от множества других программ типа систем СУБД, являются редактор правил, поддерживающий базу знаний задачи, и процедуры логического вывода, основанные на введенных правилах.

Работа с базой знаний заключается в задании системы правил. В наибольшем числе экспертных систем правила задаются, как и в обычных языках программирования, т. е. через операторы IF—THEN. У каждого правила имеется свой номер. Кроме того, в базе знаний сохраняются все особенности БД, например, поиск правила, использующего заданные переменные, добавление правил, редактирование правил и т. п.

Рассмотрим пример выполнения процедуры задания правил. Для этого мы должны войти в специальный раздел «Редактор правил», в котором мы можем редактировать и опре-

База данных	База знаний	Просмотр данных	Лог. Вывод	Исполнение	Программы
-------------	-------------	-----------------	------------	------------	-----------

Меню:

Ссылки

Лед.

Пред.

Файл

Поле

Редактор

Выход в БД

Ссылки на данные

Текстual: U2

Файл БД: Circuit1.dbf

Поле:

VOLTAGE

Переменные Базы Знаний

U1

U2

U3

U4

U5

...

Рис. 2.31. Пример работы с экспертной системой на ПЭВМ: режим установления соответствия между переменными базы знаний и базы данных

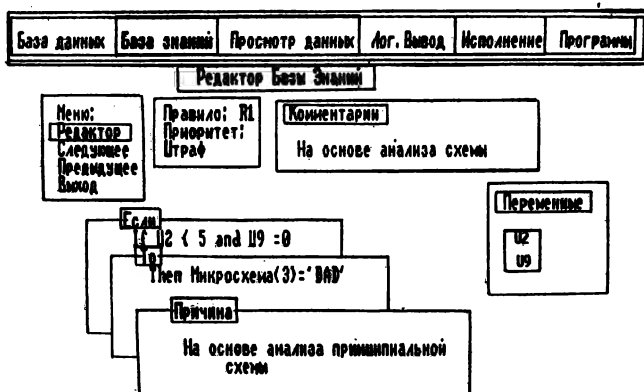


Рис. 2.32. Редактор базы знаний в экспертной системе ПЭВМ

делять новые правила (рис. 2.32). Зададим номер правила, например R1. На экране появляется специальное окно условия «Если», в котором мы теперь можем определять любые операции над переменными: сравнение, математические преобразования и т. п. Введем первое правило:

If $U(2) < 5$ and $U(9) = 0$

После возврата каретки появляется окно «То», в котором мы напишем

Then Микросхема (3) = 'Bad'

Теперь появится окно, в котором мы можем записать причину данного вывода. Запишем туда «На основе анализа принципиальной схемы». Правило задано. Задав таким же способом остальные правила, сформируем базу знаний нашей задачи.

Теперь можем в любой момент найти правило по номеру, по признаку того, что данное правило использует переменную U2, стереть его или изменить на новое точно так же, как это было бы сделано с помощью текстового редактора. В редакторе правил можно также задать вероятность, с которой выполняется заданное правило. Обычно это делается с использованием шкалы оценок от 1 до 100. Например, задав правило, можно указать, что по данной шкале уровень его достоверности равен 85. Современные экспертные системы на ПЭВМ, имеющей 640 Кбайт памяти, могут содержать наборы до 5000 правил.

При выполнении процедуры логических выводов пользователь входит в специальный раздел меню и указывает основные цели такого вывода, способ (прямой или обратный), а также в какой форме представить полученный результат: вывести на экран, записать в файл или в специальное поле БД, передать программе пользователя и т. п. Пример выполнения

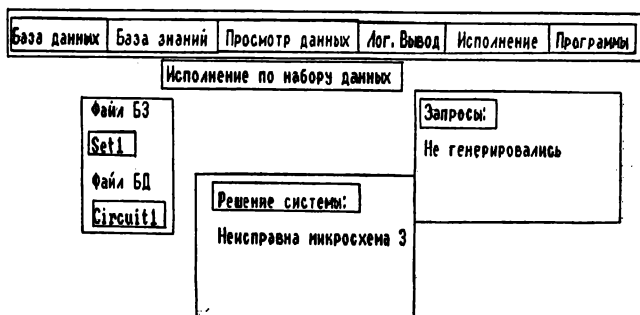


Рис. 2.33. Пример выполнения процедуры логического вывода в экспертной системе на ПЭВМ

логического вывода в подобной системе приведен на рис. 2.33. Пользователь указывает системе набор логических правил, файл БД, содержащий необходимые переменные, и получает решение о неисправностях схемы. Такая система может оказаться неоценимой при массовом контроле промышленных изделий.

Как и при работе с БД, в экспертные системы включен свой язык программирования, с помощью которого можно написать специализированную программу, содержащую функции задания переменных, правил, циклы и условные переходы. В рамках этой программы предусмотрены обычно две ключевые процедуры: синтаксический анализатор, анализирующий набор правил, и процедура выводов, реализующая прямой или обратный вывод по заданному набору правил. Функции этого языка могут быть также вызваны из программы пользователя, написанной на обычном языке высокого уровня, например на Си. Это дает пользователю широкие возможности. Например, работа с экспертной системой может закончиться созданием набора правил, записанного в файл RULESET. С помощью экспертной оболочки пользователь дорабатывает систему правил до тех пор, пока не убеждается, что в определенных ситуациях система дает правильные ответы. Тогда пользователь из своей программы может сделать вызовы процедур типа

CALL INFERENCE („RULESET“, RESULT)

и получить решение поставленной задачи в специальном образом определенном массиве RESULT. Теперь значения этого массива можно анализировать прикладной программой, вызывать другие процедуры и т. п.

Таким образом, описанную нами простейшую экспертную систему можно рассматривать как надстройку над БД, предоставляющую возможности гибкого редактирования системы

правил, организации запросов и выполнения процедур логического вывода.

На ПЭВМ, совместимых с IBM PC, распространены экспертные системы MicroExpert фирмы MicroExpert Systems, EXSYS (Exsys Inc.), Guru (Micro Data Base Systems), M1, Xi и др.

Подведем некоторые итоги. Если забыть про громкие слова типа «Искусственный интеллект», «Экспертная система», то практически мы имеем дело с удобной надстройкой, позволяющей ускорить программирование ситуаций, в которых необходимо принять решение, основанное на наборе взаимосвязанных логических правил. Интеллекта в такой системе не больше, чем в программе управления базой данных, извлекающей информацию из файла. Однако то, что эта информация извлекается с высокой скоростью, почему-то никого не удивляет, в то время как выполнение компьютером логических выводов неподготовленного человека приводит в восторг. В то же время нет никакого сомнения в том, что в ближайшее время экспертные системы на ПЭВМ начнут широко применяться для обеспечения рутинной части работы: диагностики, «подсказок» стандартных технических решений и т. п.

2.9. ИНТЕГРИРОВАННЫЕ СИСТЕМЫ

Математическое обеспечение, о котором пойдет речь в этом параграфе, в настоящее время является наиболее известным и распространенным на ПЭВМ, так как оно значительно облегчает труд человека, связанного с поиском информации, подготовкой текстов, графики, простыми расчетами и подбором различных вариантов, организаций, планированием, словом, с переработкой деловой информации. Эти системы не менее важны, чем рассмотренные нами пакеты для решения специфических инженерных задач, так как удельный вес организационной деятельности всякого рода, ложащейся на инженера или научного сотрудника, чрезвычайно велик: это подготовка отчетов, описаний, статей, документации, составление заказов на оборудование и т. п. Все эти задачи взаимосвязаны: в текст необходимо вставлять числовые таблицы, данные для которых получены из БД, производить простые пересчеты полученных чисел, на основе получаемых данных строить графики и т. п. Число различных пакетов для решения перечисленных задач составляет, по крайней мере, несколько тысяч.

Большинство пакетов для информационного обеспечения в настоящее время строится по принципу так называемых интегрированных систем. В интегрированной системе все необходимые средства объединены в рамках одного пакета, так что по мере работы не возникает необходимости в использовании другого математического обеспечения. Мы уже рассматривали практически все необходимые компоненты организационного математического обеспечения ПЭВМ:

- удобный интерфейс к командам ОС,
- текстовый редактор,
- система управления,
- машинная графика,
- электронные таблицы.

Нам осталось рассмотреть только принципы работы с так называемыми электронными таблицами (spreadsheets). Электронные таблицы почти ничем не отличаются от обычного калькулятора, с той только разницей, что простые

арифметические операции в ней производятся не над одним числом, а над набором чисел, введенных в специальную таблицу. Действительно, ПЭВМ ничего не стоит посчитать сумму или произведение сразу некоторого массива чисел, соответствующих, например, столбцу числовой таблицы. Этот простой прием, однако, относится к числу методов, про которые говорят «просто, но сердито». Для того чтобы читатель понял, о чем идет речь, рассмотрим простой и типичный пример.

Лаборатории или отделу необходимо (как всегда срочно) подготовить заказ на электронные детали, которые будут использоваться в следующем квартале. Руководитель представляет себе, какие детали и в каком количестве будут необходимы, но ему срочно требуется составить калькуляцию: сколько и каких деталей должно быть закуплено, если на эту закупку выделена определенная сумма, скажем 5000 руб.

Обычно заказ оборудования является весьма трудоемкой операцией: необходимо вооружиться калькулятором, выписать на лист бумаги наименования деталей, их стоимость, найти полную сумму для закупки каждого вида деталей, неоднократно перепроверить все вычисления, сложить результаты. При этом может оказаться, что при данном комплекте средства перерасходованы или недорасходованы. В этих случаях придется начать процедуру уменьшения количества дорогих модулей или включения дополнительных узлов, снова пересчитать произведения и суммы, перепроверить их; сложить, начать новую итерацию и т. д. Авторам приходилось заниматься аналогичной работой и они могут заверить читателей, что описанная выше процедура весьма неприятна. Однажды в наши вычисления вкралась незаметная ошибка, связанная с тем, что при сложении было пропущено одно число, и мы несколько часов не могли понять, почему же в результате получается абсурдный результат. Для составления заказа из сорока-пятидесяти наименований приходится обычно потратить весь рабочий день, а затем еще и привести результаты в нормальный вид: напечатать без помарок, оформить и т. д.

При использовании электронной таблицы этот процесс выглядит следующим образом. На экране возникает изображение рамки со стандартными позициями, которые пронумерованы по горизонтали буквами, а по вертикали цифрами. Каждая ячейка таблицы, таким образом, имеет свое обозначение, например A1, B5 и т. п. Каждое перемещение курсора заставляет маркер перепрыгивать в соответствующие клетки таблицы. Ячейки могут иметь различные размеры, и мы можем записывать в них текстовую или числовую информацию.

В первый столбец начнем записывать наименование деталей. Это делается практически так же, как и с помощью текстового редактора: можно исправлять ошибки, стирать текст и т. п. В следующий столбец сразу будем записывать стоимость детали, т. е. практически повторим имеющуюся в каталоге спецификацию, а в третий столбец запишем желательное количество деталей данного типа. Теперь воспользуемся вычислительными возможностями таблицы: потребуем, чтобы в каждой клетке таблицы, находящейся в столбце D, было записано произведение числа деталей на стоимость. Это можно сделать, записав для каждой ячейки (например, пятой) следующее выражение:

+ B5*C5

Знак «+» указывает, что в данной ячейке записано не символьное, а математическое выражение. Как только мы ввели формулу в ячейку таблицы, в ней появляется подсчитанное выражение. Теперь нас интересует общая стоимость всех деталей. У нас свободна ячейка таблицы E1. Запишем в нее следующее выражение:

@ SUM (D1..D10)

Это означает, что в ячейке E1 записывается сумма всех стоимостей из ячеек D1—D10. Теперь мы можем захотеть оценить перерасход или остающуюся сумму при данном комплекте заказываемых деталей. Для этого в ячейку E3 введем формулу + 5000—E1. Электронная таблица задана! Теперь все подсчеты в ней происходят автоматически: мы мгновенно пересчитываем и сумму и перерасход в зависимости от количества заказываемых деталей. Процедура комплектации теперь выглядит следующим образом:

Р3: 5000-Р1

Таблица	Диапазон	Копир.	Диск.	Файл	Печать	Граф.	Данные	Система	Выход
Общая	Вставка	Гореть	Строка	Убр.	Загол.	Имя	Статус	Страница	
	A	B	C	D	E	F	G	H	
	Наименование	Цена	Кол-во	Всего	Полная стоимость				
1	K572	7	30	210	5598.75				
2	K174	4.8	50	240					
3	K594PA1	10	25	250	-598.75				
4	K1804	6.4	45	288					
5	K531	3	35	105					
6	K1533	9.8	150	1470					
7	K140	3.35	45	150.75					
8	KP541	39	40	1560					
9	K224	3	25	75					
10	K589	50	25	1250					

а)

Р3: 5000-Р1

Таблица	Диапазон	Копир.	Диск.	Файл	Печать	Граф.	Данные	Система	Выход
Общая	Вставка	Гореть	Строка	Убр.	Загол.	Имя	Статус	Страница	
	A	B	C	D	E	F	G	H	
	Наименование	Цена	Кол-во	Всего	Полная стоимость				
1	K572	7	30	210	4998.75				
2	K174	4.8	50	240					
3	K594PA1	10	25	250	1.25				
4	K1804	6.4	45	288					
5	K531	3	35	105					
6	K1533	9.8	150	1470					
7	K140	3.35	45	150.75					
8	KP541	39	40	1560					
9	K224	3	25	75					
10	K589	50	13	650					

б)

Рис. 2.34. Пример работы с системой обработки электронных таблиц ПЭВМ:

а) обработка заказа на интегральные схемы; б) подбор варианта заказа, удовлетворяющего ограничению на его стоимость

Предположим, что, как показано на рис. 2.34, а, мы заказали слишком много деталей и перерасходовали почти 600 руб. Теперь достаточно, окинув одним взглядом введенный в ПЭВМ комплект, изменить количество любых деталей и мгновенно пересчитать все цены. Например, уменьшим количество дорогих микросхем K589 с 25 до 13 штук (рис. 2.34, б). Получим пересчитанные значения: мы уложились в допустимые деньги. Теперь мы можем увеличить число одних деталей, уменьшив другие, и быстро подобрать желаемый результат. Совсем не обязательно нести полученный список машинистке: достаточно дать одну команду и вся спецификация, имеющаяся на экране, попадет в текстовый редактор. Теперь можно написать необходимую «шапку» и заказ готов. Он получился по крайней мере в 10 раз быстрее, чем при его составлении традиционным путем.

В каком-то смысле идея электронной таблицы близка программам интерактивных вычислений типа TK⁺SOLVER (см. § 2.3). Одно из основных применений электронных таблиц—подбор различных вариантов в рамках системы, имеющей большое число параметров, связанных жесткими условиями. При этом пользователь может варьировать числовые значения этих параметров и подбирать необходимое решение.

Мы рассмотрели только простейшие функции работы с электронной таблицей. В действительности, мы можем копировать в нее данные из БД, стирать и перемещать любые области таблицы, задавать вычисления по сложным математическим формулам и т. п. Эти возможности открывают широкий простор для самых разнообразных приложений: с помощью электронной таблицы можно проводить сложные финансовые расчеты, находить оптимальные варианты с помощью математических алгоритмов, обрабатывать графики и т. п. Рассмотрим их более подробно для того, чтобы читатель представил себе типичную структуру команд электронной таблицы.

При вызове программы работы с электронной таблицей пользователь попадает в меню (верхняя строка на рисунках), состоящее из следующих основных компонентов:

- установка параметров таблицы,
- задание рабочей области таблицы,
- копирование и перемещение данных в таблице,
- операция с файлами данных,
- операции вывода данных на печать,
- меню для построения графиков,
- операции с данными,
- системные функции (просмотр каталогов, выполнение команд ДОС и др.).

В режиме установки параметров электронной таблицы пользователь может выполнять следующий набор операций: задать максимальный и минимальный размер столбцов и колонок таблицы, вставить в заданные ячейки таблицы пустые клетки, стереть область памяти электронной таблицы, создать способ вывода информации на экран. Например, можно разделить экран ПЭВМ на несколько окон и одновременно визуализировать в них отделенные друг от друга участки памяти, например колонку А, В и колонки R, S, T, в исходном состоянии находящиеся за полем экрана.

В режиме задания рабочей области таблицы можно выделить участок памяти, например один столбец таблицы, и указать его границы: D1..D45. Теперь выделенной области можно присвоить свое имя и затем при выполнении вычислений указывать это имя как ссылку. Например, вместо выражения @SUM(D1..D45) достаточно будет написать @SUM(COST). В данном разделе меню можно также задать формат представления числовых данных выделенной области, защитить выделенную область от записи, транспонировать (превратить столбец в колонку), стереть, скопировать и т. п.

Операции копирования и перемещения данных в таблице позволяют выделить произвольную часть памяти и перенести имеющиеся в ней данные в другую область таблицы.

Операции с файлами данных позволяют считать и записать результирующую таблицу в файлы на диске. При этом предусмотрено множество сервисных возможностей: можно считывать или записывать только часть таблицы, объединять данные, содержащиеся в нескольких файлах данных, просмотреть каталог.

Программы для обработки электронных таблиц чаще всего содержат графические средства: можно указать диапазон ячеек, по которым строить график, тип графика и получить результаты на экране. Эта возможность весьма важна, так как человек лучше воспринимает графические данные, чем набор чисел. Так, в рамках нашего примера можно построить распределение стоимости деталей в зависимости от их наименования. Для этого войдем в специальное меню построения графика, укажем столбец таблицы, соответствующий оси ординат (A1—A10) и значения функции стоимости (D1—D10). Теперь можно выбрать тип графика: линейный, гистограмму, круговую диаграмму (см. § 2.4), цвет, запомнить графический файл на диске

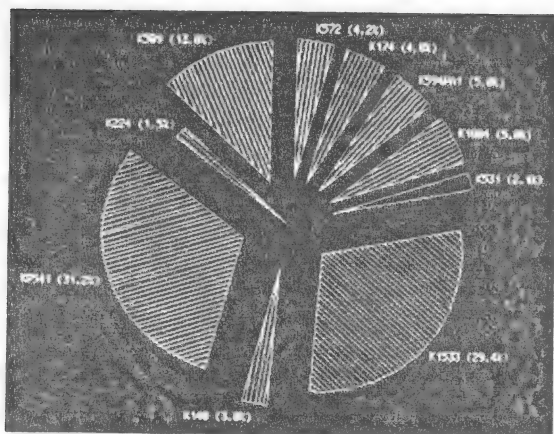


Рис. 2.35. Пример представления данных электронной таблицы в виде круговой диаграммы: распределение стоимости закупаемых микросхем

и т. п. Так, на диаграмме рис. 2.35 можно сразу увидеть, что наибольшая доля стоимости приходится на микросхемы серии КР541, К1533, а наименьшая на микросхемы К224, что, надо сказать, не совсем очевидно при взгляде на числа в электронной таблице.

Наконец, в меню операций с данными включены различные сервисные операции: сортировка по числовым значениям данных в таблице, поиск чисел или текстов, соответствующих набору логических условий, вычисление статистических характеристик (среднее, дисперсия, регрессия), матричные операции и т. п. Эта часть электронной таблицы отчасти заменяет функции БД.

На ПЭВМ в настоящее время используется несколько десятков программ для работы с электронными таблицами. Наиболее известными среди них являются пакеты Lotus 123 фирмы Lotus Development Corp., SuperCalc, Perfect Calc, VisiCalc и др.

Для полного счастья недостает только возможности считывания в электронную таблицу информации, записанной в БД, и обратной операции — запоминания в БД результатов, полученных с помощью электронной таблицы. В ряде систем такая операция возможна. Для этого необходимо выполнить следующую простую операцию: указать системе, каким полям в БД соответствуют значения ячеек электронной таблицы. Например, предположим, что подготовленный нами заказ на детали уже выполнен. Тогда для записи информации из электронной таблицы в базу данных проведем следующие операции:

войдем в специальное меню, обеспечивающее связь электронной таблицы и БД,

зададим диапазон ячеек таблицы, например А1..А10,

укажем поле БД, в которое необходимо загрузить информацию, например «Наименование»,

проделаем аналогичную операцию для других столбцов таблицы и полей БД,

дадим команду на перенос информации в любую сторону.

Теперь открываются различные возможности для извлечения информации из БД (см. § 2.7): при выполнении процедуры поиска в БД по заданному условию (например, по дате получения микросхем) можно автоматически поместить выбранную информацию в ячейки таблицы, посчитать все, что нам

хочется, «положить» информацию обратно в БД и т. д. В более простых системах обычно предусмотрена возможность трансляции файлов БД в файлы для электронной таблицы.

Как мы уже упоминали, достаточно одной команды, для того чтобы перенести содержимое электронной таблицы или выборку информации из БД в текстовый редактор (и обратно). Теперь мы можем дополнить сухие численные данные разъяснениями, необходимыми комментариями и описаниями и подготовить все необходимые документы.

Добавим к этому и функции надстройки над операционной системой: возможность просмотра каталогов файлов, их копирования, стирания, смены текущего каталога и т. д. Читателю должно быть понятно, что если за ПЭВМ сидит экономист или бухгалтер, то у него практически не возникает необходимости выходить из интегрированной системы и пользоваться другим математическим обеспечением.

Прикладное математическое обеспечение, объединяющее в себе функции перечисленных программ, т. е. обеспечивающее одновременное использование в рамках одной и той же системы графики, текстов, таблиц, БД, называют интегрированными системами. Из интегрированных систем наиболее известны пакеты Symphony (Lotus Development), FrameWork (Ashton-Tate), KnowledgeMan (Micro Data Base Systems). Среди отечественных разработок особо следует отметить интегрированную систему «Мастер» (Е. Веселов, ВЦ АН СССР).

Системы Symphony, KnowledgeMan позволяют рассматривать элементы записей в БД, тексты и ячейки электронной таблицы как единое целое: сохраняя на экране одни и те же данные, можно просто как бы менять на них точку зрения, переходя из электронной таблицы в редактор и т. п. В системах предусмотрен многооконный интерфейс: при работе пользователь, переходя из одного окна в другое, меняет «среду», выполняет операции поиска в БД, использует редактор текста и т. д.

Необычно построение системы FrameWork. Все компоненты системы рассматриваются как фреймы—упорядоченные наборы информации, причем пользователь может дробить фреймы на набор фреймов более низкого порядка и т. п. Например, в поле текста можно вставить метку, указывающую на то, что в этом месте должен располагаться участок данных из электронной таблицы, график и пр. Эти фреймы можно будет описать позже, а затем «собирать» все фреймы в единый документ. В системе имеется свой язык программирования Fged.

В интегрированную систему «Мастер» включены текстовый процессор «Лексикон» (см. гл. I), процессор электронных таблиц, БД, графический процессор. Кроме того, существует возможность писать программы на языке «Мастер». В этот язык включены функции работы с ячейками электронных таблиц, окнами экрана (рамками), функциями, определенными внутри подсистем. Таким образом, с помощью системы «Мастер» существует возможность сформировать свою версию интегрированной системы, поддерживающую необходимые в данном конкретном приложении функции. Система «Мастер» является примером разумно построенной интегрированной системы, работа с ней естественна и удобна. С помощью этой системы в настоящее время ведется разработка различных информационных систем для использования в народном хозяйстве. Мы настоятельно рекомендуем читателю приобрести систему «Мастер».

Более сложный вид программных продуктов — операционные оболочки. Если в интегрированных системах набор прикладных программ, объединенных в единое целое, определен заранее, то с помощью операционных оболочек можно создать свой комплект взаимодействующих прикладных систем. Например, экспериментатору может понадобиться система, в которой кроме средств графики и обработки на электронных таблицах необходимо также производить считывание экспериментальных данных с какого-нибудь прибора. Тогда в пакет следует включить подсистему сбора информации. Собранную информацию необходимо научиться передавать в программы обработки для анализа. Для этого в системе должны быть предусмотрены специальные про-

цедуры, с помощью которых указывается, какие именно данные необходимо брать из каждой прикладной системы, в каком формате они записаны (двоичном, символьном и т. п.), что с ними надо делать и пр. Наиболее распространенными прикладными системами такого рода на ПЭВМ являются оболочки Top View (IBM), GEM (Digital Research), MS-Windows (Microsoft) и др. Пользователь может сам определить вид диалога, при помощи которого он будет обращаться к нестандартному компоненту программного обеспечения и способ его вызова. При этом экран разбивается на различные окна, в которых содержится необходимая информация. К сожалению, в настоящее время действительно универсальной системы такого рода не существует.

Таким образом, если подвести итоги, то в настоящее время уже сложилась технология разработки интегрированных систем для обработки информации, создающая значительные удобства в сфере управления, подготовки документов, анализа решений. Развитие этой области программного обеспечения в ближайшие годы изменит процедуру взаимодействия пользователя с ПЭВМ, так как появятся действительно универсальные операционные оболочки, позволяющие пользователю интегрировать в единую систему необходимый ему комплект программ и пользоваться единообразными процедурами диалога.

Глава 3. ОПЕРАЦИОННАЯ СИСТЕМА, СТРУКТУРА И ТЕСТИРОВАНИЕ ПЕРСОНАЛЬНОЙ ЭВМ

Читатель, который прочел первую главу, уже обладает некоторыми познаниями в операционной системе и структуре ПЭВМ. К сожалению, современные компьютеры еще очень далеки как от идеала далекого будущего (известный советский математик Ю. И. Манин считает, что компьютеры далекого будущего будут похожи на прозрачный лист стекла, положив который на написанный на бумаге текст задачи можно будет сразу увидеть ее решение), так и от постепенно становящегося реальным «черного ящика», снабженного диалоговыми прикладными программами, общаться с которым на языке операционной системы придется лишь системным программистам. Поэтому если читатель этой книги желает грамотно работать с ПЭВМ, ему придется изучить эту главу. Конечно, как и в гл. 1 в ней содержится лишь очень ограниченный набор сведений, но пользователю ПЭВМ, по мнению авторов, этого набора будет вполне достаточно для квалифицированной работы.

3.1. ОПЕРАЦИОННАЯ СИСТЕМА ДОС, ЕЕ СТРУКТУРА И ВОЗМОЖНОСТИ

В гл. 1 мы рассмотрели несколько наиболее распространенных команд ДОС, с помощью которых пользователь может просматривать каталог, стирать и копировать файлы, создавать и уничтожать каталоги. Кроме этих команд, в операционной системе ДОС предусмотрен ряд дополнительных, которые используются обычно реже, но предоставляют значительные

удобства. Эти команды мы и рассмотрим. Для удобства изложения разделим их на несколько групп:

- команды работы с каталогом,
- команды работы с файлами,
- команды работы с дисками,
- системные команды.

Команды работы с каталогом. Эти команды частично уже были рассмотрены. Напомним их (см. 1.2):

- DIR — позволяет просмотреть каталог файлов,
- MKDIR (MD) — создает новый каталог,
- RMDIR (RD) — уничтожает пустой каталог,
- CHDIR (CD) — позволяет войти в каталог.

Читатель помнит из содержания гл. 1, что ДОС позволяет организовать иерархическую структуру файлов: на одном диске можно создать много различных каталогов, причем, ссылаясь на файл, необходимо указать полный путь к нему по дереву файлов. Если мы находимся в корневом каталоге и даем команду DIR, то «входы» в другие каталоги на диске индицируются в списке файлов как объекты, имеющие расширение <DIR>. В ДОС предусмотрена специальная команда, позволяющая сразу просмотреть структуру дерева файлов. Эта команда называется TREE (дерево). Если дать команду TREE, то на экране появится список каталогов на диске и подкаталогов, выходящих из этих каталогов. Например, если вспомнить пример, приведенный в § 1.2, то, дав эту команду, увидим примерно следующее сообщение:

СПИСОК КАТАЛОГОВ НА ДИСКЕ

Каталог \CC

Подкаталоги: отсутствуют

Каталог \BASIC

Подкаталоги: отсутствуют

Каталог \TEXT

Подкаталоги: \PETROV

\IVANOV

Каталог: \TEXT\PETROV

Подкаталоги: \GAMES

\SIDOROV

Каталог: \TEXT\PETROV\GAMES

Подкаталоги: отсутствуют

Каталог: \TEXT\PETROV\SIDOROV

Подкаталоги: отсутствуют

Каталог: \TEXT\IVANOV

Подкаталоги: отсутствуют

Если команду TREE записать как TREE/F, то кроме списка каталогов, система покажет также и все файлы, находящиеся в них.

Читатель помнит, что для того чтобы пользователю Иванову, работающему в своем каталоге, вызвать компилятор языка Бейсик BASCOM, находящийся в каталоге BASIC, необходимо дать команду \BASIC\BASCOM, иначе ДОС не найдет этот файл. Это представляет большие неудобства: с одной стороны хорошо, что файлы, находящиеся в различных каталогах, «не видны» друг для друга и не будут путаться, но, с другой стороны, очень неудобно каждый раз указывать системе полный путь по дереву файлов при вызове компилятора, редактора, даже некоторых команд самой ДОС. Для того чтобы решить эту проблему, было бы хорошо иметь возможность сделать некоторые ветви дерева файлов «прозрачными» для всех пользователей. Лучше всего сделать прозрачными те каталоги, в которых записаны системные программы: команды ДОС, компиляторы, редакторы, прикладные пакеты и т. п. Эту операцию можно сделать с помощью специальной команды ДОС PATH (путь). Смысл этой команды в том, что система будет искать выполнимые файлы (с расширениями .COM, .EXE, .BAT) только в тех каталогах, которые указаны в данной команде. Так, для нашего примера разумно дать следующую команду:

```
PATH C:\;C:\CC;C:\BASIC;C:\TEXT
```

Теперь корневой каталог и каталоги CC, BASIC и TEXT становятся прозрачными для всех пользователей данной машины и любой из них может смело вызывать программы, записанные в этих каталогах, без указания полного имени файла.

Наконец, еще одна команда, относящаяся к группе команд работы с каталогами, записывается следующим образом: SUBST (от английского слова substitute — подстановка). Смысл этой команды в том, что какому-то из каталогов можно присвоить заданное логическое имя, как будто мы работаем с дополнительным диском. Например, команда

```
SUBST C: \TEXT\PETROV\SIDOROV
```

присвоит каталогу Сидорова обозначение d: и теперь мы можем, например, дать команду DIR d: вместо команды

```
DIR C:\TEXT\PETROV\SIDOROV
```

Применять команду SUBST имеет смысл только в довольно специфических случаях, например, когда пользователь Иванов вынужден очень часто обращаться к файлам Сидорова и каждый раз указывать их полные имена затруднительно. Заметим, что в качестве логического имени в этой команде можно использовать только свободные допустимые буквы обозначения дисков для данной конфигурации ПЭВМ.

Команды работы с файлами. Мы рассмотрели в гл. 1 три команды:

COPY — копирует файлы,

DEL — стирает файлы,

TYPE — выводит содержимое файлов на экран:

В ДОС версии 3.2 имеется также удобная команда для копиро-

вания файлов: XCOPY. Она записывается следующим образом:

XCOPY файл1 файл2 [/d:дата] [/e] [/p] [/s] [/v]

где параметры, указанные в скобках, являются вспомогательными. Если дать команду XCOPY без этих параметров, то она ничем не отличается от команды COPY. Рассмотрим значение наиболее важных дополнительных параметров команды XCOPY:

/d — копирует файлы, созданные только после определенной даты;

/e — копирует файлы из всех каталогов, выходящих из данного;

/p — при копировании каждого файла пользователю задается вопрос «Скопировать файл? (Y/N)». Если ответить «Y», файл будет скопирован;

/v — автоматическая проверка правильности копирования. Например, команда

XCOPY a: b: /e

полностью скопирует один гибкий диск на другой, сохранив всю файловую структуру;

XCOPY *.txt a: /p/v

скопирует все текстовые файлы на гибкий диск, спрашивая у пользователя разрешения на копирование каждого файла и автоматически проверяя правильность копирования.

Кроме работы с этими командами, часто приходится выполнять ряд других операций, например изменять имена файлов. Этого можно добиться командой RENAME (или просто REN). Например, команда

REN LEXICON.EXE REDACTOR.EXE

переименует файл LEXICON в файл REDACTOR. Для того чтобы напечатать файл на принтере, предусмотрена команда PRINT. У этой команды имеется около десятка различных параметров, рассматривать которые мы здесь не будем. В простейшем случае достаточно просто написать

PRINT DOCUMENT.TXT

и файл DOCUMENT будет напечатан.

Команда COMP (сравнение, от английского слова COMPARE) позволяет сравнить содержание различных файлов. Эту команду имеет смысл применять после операции копирования файлов для проверки правильности копирования или в тех случаях, когда у пользователя появилось несколько версий файлов, имеющих одинаковый размер и необходимо проверить их идентичность. Например, если Сидоров, как было описано в примере из § 1.2, несколько дней назад уже приносил на ПЭВМ свои файлы с текстами, а сегодня хочет проверить, не изменялись ли они с тех пор на его дискете, то он должен вставить ее в компьютер и дать следующие команды:

CD\TEXT\PETROV\SIDOROV

COMP *.TXT A: *.TXT

При этом будет произведено побайтовое сравнение всех текстовых файлов, записанных на фиксированном и на гибком дисках имеющих одинаковые имена. Если файлы идентичны, то компьютер выдаст сообщение «Compare OK», в противном случае появится надпись «Ошибка сравнения по адресу...».

Предположим, что Сидоров увидел, что его файлы на фиксированном диске неизвестно кем изменены чьей-то беспощадной рукой: в них вместо текстов записана какая-то чепуха. Для того чтобы исключить случайную перезапись своих файлов, в DOS предусмотрены возможности защиты файла от несанкционированной записи и при необходимости снятия такой защиты. Это делается с помощью команды ATTRIB. Например, команда

```
ATTRIB +R DOCUMENT.TXT
```

защищает файл от записи, а команда

```
ATTRIB -R DOCUMENT.TXT
```

убирает защиту.

В тех случаях, когда диск почему-то испортился и на нем появились плохие сектора, отдельные файлы или целый каталог его можно попытаться восстановить, используя команду RECOVER. Эта команда произведет чтение диска по секторам и пропустит данные, записанные в плохом секторе. Для файлов содержащих коды, такой пропуск обычно «смертелен», но текстовые файлы иногда еще можно восстановить. Например, если в файле DOCUMENT.TXT появился плохой сектор, то команда

```
RECOVER DOCUMENT.TXT
```

восстановит его неиспорченную часть. Испорченная часть при этом будет утеряна.

При копировании большого числа файлов на гибкие диски или в какой-нибудь служебный каталог можно использовать команду BACKUP. Эта команда отличается от команды COPY: файлы, скопированные при помощи команды BACKUP, содержат специальную контрольную информацию и для использования их нужно восстанавливать специальной командой RESTORE. Преимущества использования такого метода копирования — в том, что пользователю не нужно заботиться о том, сколько именно файлов поместится на гибком диске: если файлы не поместятся на нем, система сама предложит вставить следующий диск в компьютер. Точно так же при восстановлении скопированных файлов в системе будет записана информация о том, на скольких дисках хранится информация. У команды BACKUP имеется несколько дополнительных параметров, с помощью которых можно копировать все файлы из подкаталогов (флажок /S), копировать только те файлы, которые были изменены после последней операции копирования (флажок /M), добавить копируемые файлы к содержимому данного диска (флажок /A), копировать только те файлы, которые были изменены после некоторой даты (флажок /D). Например, для

того чтобы скопировать весь фиксированный диск на гибкие диски, необходимо дать команду

BACKUP C: *.* A: /S

а для того чтобы провести обратную операцию — дать команду

RESTORE A: C: *.* /S

Команды работы с дисками. Наиболее частой операцией при работе с дисками является форматирование гибких дисков. Процедура форматирования необходима при подготовке новой дискеты к использованию ДОС. Для того чтобы провести форматирование нового гибкого диска, обычно достаточно дать команду

FORMAT A:

При этом на экране появится надпись: «Вставьте диск в дисковод, дайте возврат каретки» и процедура форматирования начнется. После ее завершения на экране появится надпись: «Формат завершен» и информация о том, сколько байтов доступно на диске. Теперь на этот диск можно копировать файлы. У команды **FORMAT** имеется ряд параметров, например можно указать тип диска (двусторонний или односторонний), можно скопировать на гибкий диск ДОС (флажок **/S**). Заинтересовавшийся читатель может найти описание этих параметров в руководстве по ДОС.

Команда **SYS** позволяет перенести операционную систему с одного диска на другой. Например, если мы хотим записать файлы ДОС с фиксированного диска на флоппи-диск, достаточно написать

SYS A:

Довольно часто возникает необходимость скопировать файлы с одного гибкого диска на другой. Для этого в ДОС имеется команда **DISKCOPY**. В ПК могут быть установлены как два дисководы для флоппи-дисков, обозначаемых **A:** и **B:**, так и только один дисковод. В любом случае для копирования дискетов необходимо дать команду

DISKCOPY A: B:

которая означает: скопировать содержимое диска **A:** на диск **B:**. Если ПК имеет только один дисковод, то ДОС будет все время сообщать пользователю: «Вставьте исходный диск», «Вставьте диск, на который копируются файлы», и произведет копирование, прочитывая фрагменты памяти с исходного диска в оперативную память ПЭВМ и затем записывая их на другой диск. Если на компьютере имеется два дисковода, то несколько раз переставлять диски не придется: система только один раз попросит Вас вставить исходный диск в дисковод **A:**, а новый диск в **B:**, и сразу же произведет всю операцию копирования.

Скопированные диски можно сравнить, проверив таким образом соответствие записанной в них информации. Для этого достаточно дать команду

DISKCOMP A: B:

Если при выполнении этой операции будут обнаружены ошибки, то на экране появится специальное сообщение.

Для того, чтобы проверить «состояние» фиксированного или гибкого диска, достаточно дать команду CHKDSK. Эта команда анализирует состояние диска, каталогов и файлов на нем и находит поврежденные сектора диска, если они есть, выдавая об этом сообщения. Наличие поврежденных секторов или испорченных файлов является грозным симптомом: в этом случае необходимо принимать срочные меры: копировать все файлы, которые еще можно сохранить, и пытаться снова форматировать диск. Команда CHKDSK для исправного диска выдает на выходе примерно следующую информацию:

Диск MYDISK создан 10 июня 1987 г. в 11:30

10600448 байт полная емкость диска
40960 байт в 3 скрытых файлах
36864 байт в 9 каталогах
8265728 байт в 494 файлах пользователя
2256896 байт доступно на диске

Наконец, команда FDISK используется при установке операционной системы на фиксированном диске. С ее помощью форматируется фиксированный диск, на нем отводится место для размещения операционной системы. Поскольку существует множество версий этой программы, для ее изучения мы отсылаем читателя к руководствам по конкретным версиям ДОС.

Системные команды. С их помощью можно выполнять разнообразные служебные функции. Например, команда CLS очищает экран ПЭВМ. Команды DATE и TIME позволяют пользователю установить текущую дату и время. Более сложна команда MODE: с ее помощью можно управлять отображением на экране, работой принтера, асинхронного адаптера. Наиболее часто используются команды изменения способа отображения. Например, команда MODE 80 переключает дисплей ПК в режим отображения 80 символов в строке, а команда MODE 40 — соответственно в режим отображения 40 символов. Если мы напомним

MODE LPT1:132,8

то тем самым мы указали системе, что хотим использовать принтер, подключенный к порту 1, в режиме 132 символа в строке и 8 строк на дюйм. Этот режим можно изменить, дав команду

MODE LPT2:80,6

что соответствует использованию принтера 2, печатанию 80 символов в строке и 6 строк на дюйм.

Другой очень важной и полезной командой является PROMPT. С помощью этой команды можно задать вид пригла-

шения операционной системы. Имеется в виду прежде всего то, что существует возможность указать в приглашении текущий каталог, в котором находится пользователь. Это означает, что пользователь Сидоров, находящийся в своем каталоге, вместо унылого приглашения C>, свидетельствующего о том, что он находится где-то в необъятных просторах фиксированного диска, может видеть разноцветное приглашение типа C:SIDOROV> или даже C:\TEXT\PETROV\SIDOROV>. Например, если мы напишем

```
PROMPT $n$g
```

то приглашением системы останется имя диска (Sn) и правая стрелка (Sg), т. е. для диска A мы увидим A>. Если написать

```
PROMPT $p$g
```

то вместо этого мы увидим имя текущего каталога на диске.

В различных версиях ДОС существует еще около десятка команд, которые мы не рассматривали. Однако читатель, который прочел § 1.2 и данный, может считать, что он вооружен практически всем необходимым для того, чтобы работать с ОС

3.2. ДИАЛоговые наДстройки наД ДОС

Ознакомившись с основными командами ДОС, можно заключить, что работа с этой ОС весьма проста. Действительно, для выполнения подавляющего большинства операций достаточно знать всего несколько ключевых команд, имеющих очень простую форму. Однако у многих пользователей возникает, на наш взгляд, ошибочное впечатление, что работа с операционной системой является весьма нудным и сложным делом: необходимо помнить имена каталогов, не путаться в обозначении файлов, указывать специальные ключи и т. п. По-видимому, именно для такой категории пользователей и были созданы первые надстройки на ДОС, позволяющие выполнять все необходимые функции работы с машиной, не зная команд ОС.

Итак, что же такое диалоговая надстройка над ОС? Обычно это некоторая диалоговая система, высвечивающая на экране ЭВМ красивые цветные меню, предлагающие пользователю выполнить те или иные функции ДОС: посмотреть содержимое дисков, стереть или скопировать файлы, создать новые директории, провести форматизацию дисков и т. д. Если пользователь не знает, что именно делает та или иная команда, к его услугам встроенная подсистема подсказок. Например, подробнейшим образом объясняются функции копирования файлов, создания каталогов. Пользователь клавишами управления курсором выбирает тип задания, затем появляются дополнительные вопросы и подсказки типа: «Вы намереваетесь стереть файл, укажите обозначение диска (A-C):», «Выберите и укажите файл, кото-

рый вы хотите стереть, двигайтесь по списку файлов стрелками управления курсором» и, наконец, «Вы уверены в том, что вы действительно хотите стереть этот файл?». Читатель, наверное, уже заметил, что прочесть подобную последовательность вопросов и подсказок интересно только при первом знакомстве с системой, а при последующем использовании такой интерактивной программы это начинает вызывать раздражение. Действительно, куда проще набрать на клавиатуре команду DEL A:TEST.EXE, чем бесконечно отвечать на вопросы. Более того, во многих надстройках реализовано большое число сервисных функций, каждая из которых вызывается нажатием на определенную клавишу или на сочетание клавиш, причем число этих сочетаний близко к числу команд ДОС. Последние, однако, имеют довольно ясную мнемонику, в отличие от странных сочетаний типа ALT-F8. Легко понять, что принцип диалогового выполнения команд ОС, доведенный до абсурда, превращает подобную надстройку в монстра, выучить команды которого сложнее, чем запомнить основные команды ДОС.

По нашему мнению, существующие диалоговые надстройки над ОС можно условно разделить на три категории: полезные (облегчающие каждодневную работу на ПЭВМ), умеренно полезные (облегчающие решение некоторых редко встречающихся задач) и бесполезные (практически усложняющие работу). Данная точка зрения, конечно, субъективна и может вызвать нарекания у пользователей, привыкших пользоваться той или иной системой и считающих ее разумной.

Одним из примеров громоздкой и, по мнению авторов, бесполезной надстройки является интегрированная система WINDOWS фирмы Microsoft. Эта система занимает почти 1 Мбайт дисковой памяти и рассчитана на преимущественное использование совместно с устройством типа «мышь». Она объединяет в себе функции просмотра файлов в каталогах, текстового редактора, калькулятора, календаря, графического редактора и позволяет подгружать ряд других систем. Из-за того, что эта система объединяет различные подсистемы и допускает обмен информацией между ними, ее также часто называют операционной оболочкой (см. § 2.9). Удобство такой оболочки вроде бы заключается в том, что пользователь может объединить несколько различных программ и организовать между ними обмен информацией. Например, проведя редактирование текста, можно передать его в программу обработки электронных таблиц, базу данных и т. д. Но об этом было сказано в гл. 2.

Пока рассмотрим только системный аспект работы с WINDOWS, так как мы неоднократно убеждались, что эта система в 90% случаев используется именно таким образом. Работая с этой системой, пользователь может двигаться курсором или с помощью «мыши» по списку файлов, смотреть содер-

жимое этих файлов, выходить в специальные командные строки, позволяющие выполнять команды ДОС, выполнять программы, редактировать текст, рисовать цветные картинки и многое другое. Работа с WINDOWS, конечно, впечатляет: в моменты ожидания (загрузка подсистемы, запись файлов) на экране появляется символ ожидания — песочные часы, при стирании файлов возникает изображение мусорной корзины, изменяются цвета фона и текста, окна, содержащие подсказки, накладываются друг на друга и т. п. По нашему мнению, символом чрезмерного эстетизма и громоздкости этой системы является подсистема, показывающая время. При вызове этой системы на экране появляются красивые часы с привычным циферблатом и движущимися стрелками... За красоту всегда надо чем-то платить. В системе WINDOWS эта плата — большое время перехода из одной системы в другую, громоздкость переключающих конструкций и большой объем необходимой памяти ЭВМ. Например, если пользователь находится в режиме просмотра файлов и хочет посмотреть содержимое другого каталога, он должен набрать примерно такую последовательность команд:

1. ALT-пробел, ALT-табулятор до тех пор, пока указатель не встанет на красивое изображение гибкого диска, символизирующее выбор ДОС в качестве необходимой подсистемы.

2. ALT-S (выбор подменю, содержащего функцию смены каталога).

3. Стрелками выбрать в красивом цветном окошке функцию Change directory, соответствующую смене текущего каталога.

4. Через секунду на экране появится еще одно цветное окошко, в котором будет написано имя текущего каталога и будет содержаться приглашение к вводу имени нового каталога. Теперь надо написать новое имя.

5. Теперь еще следует подтвердить, что введенное имя действительно нас устраивает. Для этого необходимо дополнительно убедиться, что курсор попал в специальное место экрана с надписью ОК и нажать возврат каретки.

Если в комплекте компьютера имеется специальное устройство типа «мышь», нажатие символов ALT-буква заменяется движением «мыши» и нажиманием на ней специальных кнопок, что много проще работы с клавиатурой. Это, конечно, ускоряет работу, но все равно скорость выполнения необходимых операций остается крайне низкой. Теперь вспомним, что в ДОС для проведенной операции необходимо лишь набрать сочетание букв CD \имя каталога, дать возврат каретки и затем набрать команду DIR. Времени на это уйдет в несколько раз меньше, чем при использовании системы WINDOWS. На рис. 3.1 показан экран при использовании этой системы.

Таким образом, читатель уже понял, что среди надстроек над ДОС бывают довольно бесполезные системы, которые только выглядят красиво, а на самом деле отнимают время пользо-

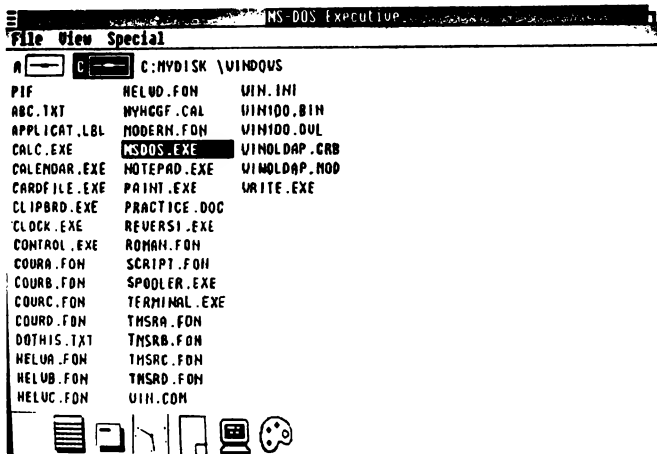


Рис. 3.1. Экран ПЭВМ при работе с интегрированной системой Windows

вателя, память на дисках и оперативную память ЭВМ. Обманчивая красота таких систем, однако, сильно воздействует на неискушенных пользователей, которые не имели практики работы на машине. Инерция мышления бывает столь сильна, что авторам приходилось наблюдать, как люди, начавшие работать с подобной надстройкой, впоследствии с трудом заставляют себя изучать команды ДОС. Хочется предостеречь от этой ошибки читателей.

Теперь несколько слов об умеренно полезных системах. Умеренно полезными мы считаем такие системы, в которых, наряду с недостатками бесполезных, реализованы некоторые функции, отсутствующие в ОС, но иногда необходимые на практике. Позиция разумного пользователя по отношению к умеренно полезным системам, по нашему мнению, должна выглядеть следующим образом: такую надстройку разумно иметь на фиксированном или гибком диске, с тем чтобы время от времени ее использовать. Умеренно полезной надстройкой над ДОС является, например, система PCTOOLS фирмы Central Point Software.

Рассмотрим примерную организацию такой системы. При ее загрузке на экране появляется меню, например типа, изображенного на рис. 3.2.а. Слева в этом меню указаны основные команды ДОС и некоторые дополнительные функции, справа в специальном окне написано объяснение того, какие именно функции выполняет та или иная команда. Движение по командам выполняется клавишами управления курсором. Например, выбрав команду DIRECTORY, система вначале запрашивает имя диска, а затем сканирует весь указанный диск и строит дерево каталогов, как это показано на рис. 3.2.б. Это очень

DIRECTORY ← → Display directory information and
COPY optionally list sub-directories.
COMPARE
SEARCH
RENAME
DELETE
VERIFY
VIEW/EDIT
UNDELETE
STATUS
MAPPING
SYSTEM INFO.
PRINT
LOCATE
FORMAT
End PC Tools

Use the UP (↑) arrow and the DOWN (↓) arrow
to select. Then press ENTER (↵).

а)

PC Tools - Directory Service Volume Label=None

Path=D:\

```

R TC-----IMC
D AUTOCAD
O RT-PRINT
T DOS
  PMSM
  LIB
  PRINT
  ECA
  BASIC
  ASSIST
  PROC
  TEXT-----PW
  
```

Use cursor arrow keys to follow the chain to the desired directory.

б)

PC Tools - Directory Service Volume Label=None

Path=C:\TEXT*.*

Name	Ext	Size	#Clu	Date	Time	Attributes
ARTICLE	DOC	15993	4	1/21/86	2:42p	Normal, Archive
BRAIN	DOC	8632	3	8/21/87	10:31a	Normal, Archive
DOCLAD	DOC	14470	4	3/26/86	5:13p	Normal, Archive
JAPAM	DOC	26674	7	6/11/87	11:51a	Normal, Archive
WTP	DOC	1043	1	12/22/87	1:13p	Normal, Archive
JAPAM	BAK	24491	6	8/21/87	10:36a	Normal, Archive
PARAM	BAK	212	1	11/04/87	1:28p	Normal, Archive
MAVES	DOC	495	1	11/03/87	1:45p	Normal, Archive
OR	DOC	221	1	11/04/87	3:28p	Normal, Archive
HUANG	RUS	1858	1	12/11/87	1:23p	Normal, Archive
FMTQ	FMT	11267	3	12/29/86	11:43a	Normal, Archive
FX80	FMT	3872	1	12/23/88	7:21p	Normal, Archive
DOCLAD	TXI	12052	3	5/13/87	11:36a	Normal, Archive
VARIANTS	DOC	1813	1	10/19/87	12:11p	Normal, Archive
SINCH	DOC	28862	6	12/04/87	4:54p	Normal, Archive
TEST	DOC	287	1	10/30/87	4:36p	Normal, Archive
OTCHEY	DOC	3853	1	10/29/87	10:53a	Normal, Archive

56 entries listed totalling 504284 bytes. 1220800 bytes available.

↑=Scroll PgUp=Page Up PgDn=Page Down Home=Top End=Bottom ESC=Exit
F7=Directory Sorting F8=Directory List Argument F10=Directory Selection

в)

Рис. 3.2. Примеры работы с надстройкой PCTOOLS
на ПЭВМ:

а) общее меню системы; б) дерево каталогов; в) список
файлов выбранного каталога

удобно, так как позволяет получить полное представление о структуре каталогов на данном диске. После выполнения этой команды тот каталог, в котором находится пользователь, будет отмечен цветом или фоном, отличным от остальных. На рис. 3.2,б этот каталог имеет имя TEXT.

Теперь достаточно дать возврат каретки для того, чтобы увидеть на экране список файлов, находящихся в данном каталоге (рис. 3.2,в). По этому списку файлов также можно двигаться с помощью клавиш управления курсором. Снова нажав «Возврат каретки», можно увидеть на экране содержимое выбранного файла. Если в системе для выполнения тех или иных функций необходимо использование дополнительных клавиш, то все они выделены специальным цветом и располагаются в окне подсказок, которое всегда присутствует в изображении. Таким образом, читатель может убедиться в явном преимуществе подобной системы: управление функциями минимизировано и сведено фактически только к управлению курсором и возврату каретки, что вполне естественно для человека, сидящего за дисплеем ПЭВМ.

В меню, показанном на рис. 3.2,а, используются практически все функции, имеющиеся в ОС: копирование и стирание файлов, создание и уничтожение каталогов, форматирование дисков. Однако их использование чересчур перегружено служебными запросами, подсказками, аналогично тому, что это было описано выше.

Теперь остановимся на том, какие именно функции, не реализованные в ДОС, обычно реализуются в подобных надстройках. Одну из них мы уже продемонстрировали на примере построения дерева каталогов и гибкого движения по этому дереву. Другой аналогичной функцией является поиск файла по имени. Имеется в виду следующее: пусть пользователю известно, что где-то в компьютере имеется файл с заданным именем, но ему неизвестно, в каком именно каталоге искать этот файл. Эта проблема может вылиться в часы поиска, особенно если дерево каталогов достаточно разветвленное. Этой функции в приведенном меню соответствует строка LOCATE.

Очень похожей на эту задачу является следующая. Пусть необходимо найти, в каком именно файле на диске имеется то или иное характерное сочетание символов или кодов. Это может быть имя, адрес, спецификация объекта в базе данных или характерный код переключения того или иного устройства. Используя надстройку над ДОС, можно решить и эту задачу: в ней имеется специальная программа, которая сканирует диск в поисках заданного сочетания символов. Найденное сочетание тут же можно заменить на другое (строка FIND). Во многих системах имеется также сервисная возможность получить информацию о версии и размерах используемой операционной

системы (SYSTEM INFO). Команда UNDELETE служит для восстановления случайно стертого файла, команда STATUS позволяет быстро изменить статус файла.

Достаточно частой операцией, которую приходится проводить практически любому пользователю ПЭВМ, хочет он этого или нет, является проверка состояния фиксированных дисков, дискет и файлов. Для этой цели в ДОС имеется команда CHKDSK. Надстройки обычно включают в себя более изощренные способы проверки дисков. Например, команда VERIFY в меню на рис. 3.2,а последовательно проверяет все секторы диска и выдает сообщения о найденных ошибках. Специальная программа (строка DISK MAP) выдает на экран изображение распределения памяти на фиксированном диске, на котором различными цветами помечены буквально все возможные ситуации: свободные и занятые области диска, скрытые и защищенные от записи файлы, испорченные сектора диска и т. п.

Проиллюстрируем поиск файла с заданным именем в такой системе. Пользователю известно, что где-то на диске ПЭВМ записан файл, называющийся README.TXT и содержащий полезную информацию. Но на диске имеется девять каталогов и просматривать все их в поисках искомого файла утомительно. Для поиска вызовем систему PCTOOLS. Двигаясь курсором по основному меню, выберем функцию LOCATE и дадим возврат каретки. В правой части экрана возникнет два окна. В первом из них будет написано: «Выберите имя диска, на котором записан файл». Поскольку мы проводим поиск на фиксированном диске, нажмем на клавишу С. Курсор передвинется в другое окно, в нем будет написано «Введите предположительное имя файла». Введем имя файла (отметим, что мы можем использовать звездочки, например, потребуем найти все файлы, называющиеся README и имеющие произвольное расширение: README.*). Дадим возврат каретки. На экране появится надпись: «Провожу поиск по каталогам» и в результате экран будет выглядеть так, как показано на рис. 3.3: система нашла три таких файла в различных каталогах и выдала всю касающуюся их информацию.

PC Tools - File Locator Service		Volume Label:None
Path:C:\		
Searching Path = C:\README.*		
README.DOC	11	1/25/87 12:10p
Searching Path = C:\TEXT\README.*		
README.TXT	9	1/25/87 12:10p
Searching Path = C:\FIG\README.*		
README.TXT	7	1/25/87 12:09p

Рис. 3.3. Экран в режиме поиска файлов на диске

У читателя может возникнуть вполне законный вопрос: чем же провинились системы, подобные только что описанной, если авторы называют их умеренно полезными? Ведь все описанные выше возможности выглядят весьма разумно, многие команды вообще незаменимы, работа с такой системой очень проста и действительно может полностью заменить команды ОС. Но такое мнение может возникнуть лишь у человека, который не работал на ПЭВМ длительное время. В чем же тот ключевой пункт, который все же позволяет нам сказать, что такая надстройка является лишь умеренно полезной для пользователя? Это — знание распределения частоты обращения пользователя к тем или иным функциям ОС и ПЭВМ при каждодневной работе. Необходимость обращения к любой из описанных выше функций при обычной работе на ПЭВМ возникает не чаще одного раза в несколько рабочих дней: не каждый день теряются или случайно стираются файлы, портится диск, изменяется версия ДОС.

Рассмотрим, какие команды чаще всего приходится выполнять человеку, сидящему за дисплеем и пишущему программу или использующему какой-то стандартный пакет программ. По-видимому, чемпионами среди используемых команд будут команды: DIR, позволяющая просмотреть текущий каталог пользователя и найти файл; CHDIR, изменяющая текущий каталог; COPY, копирующая файлы; DEL, стирающая файлы. Действительно, пользователь всегда работает с файлами и естественно, что все названные нами команды сводятся к файловой системе: необходимо найти, стереть, скопировать тот или иной файл, узнать его размер и т. п. Именно при работе с файлами проявляется та неприятная черта ДОС, что в ней необходимо полностью указывать имена каталогов, файлов и их расширений, что при частом выполнении операций с файлами из досадной необходимости быстро перерастает в раздражающую и неприятную черту системы ДОС.

Таким образом, если исходить из реальных, практических потребностей пользователя ПЭВМ, полезной надстройкой над ДОС можно считать ту, которая максимально облегчает прежде всего функции работы с файлами, сохранив возможности выполнять остальные команды ДОС и дополнительные сервисные команды.

Сразу заметим, что на момент написания книги авторам не известна ни одна надстройка над ОС для ПЭВМ, полностью удовлетворяющая всем высказанным требованиям. В то же время, существует много надстроек над ДОС, действительно удобных при работе с файлами и сохраняющих возможности реализации других команд ДОС. Сегодня именно эти надстройки подходят под нашу классификацию как действительно полезные. Такие надстройки часто также называют файловыми

мониторами или, на английском языке, File Manager. Из подобных систем, разработанных для машин, совместимых с моделью IBM PC, прежде всего можно назвать системы CAT, Path-Finder, dirWorks, DirsCat, Qtools, Scout, ShrtCut, Xtree, ALPS, SNAP, Norton Commander, DirPlus, Power Toc, Stillwaters, а также ряд отечественных разработок, из которых наиболее известны разработки ВЦ АН СССР и Таллинского политехнического института. Поскольку все эти системы имеют различные управляющие команды и различную структуру, далее опишем некоторую среднюю типичную систему, которая позволит читателю получить представление о характере работы с полезной надстройкой.

Пользователь, работающий с такой надстройкой, как правило, должен иметь возможность не выходить из нее, разрабатывая и запуская свои программы или пакеты. Таким образом, надстройка начинает играть роль ОС: после окончания работы любой программы или редактора текстов вместо возвращения в ОС происходит возвращение в тело надстройки, сразу же показывающей пользователю список текущего каталога файлов (рис. 3.4). Одна строка экрана содержит всю необходимую информацию о каком-либо файле или каталоге. При этом первая колонка символов соответствует имени файла, вторая — расширению (для каталога предусмотрено стандартное расширение типа DIR), третья — размеру, четвертая и пятая — дате и времени создания файла, шестая — атрибутам файла (скрытый или нет, защищен ли от записи). Движение по списку файлов происходит при помощи клавиш управления курсором, при этом курсор помечает, на каком именно файле в данный момент находится система. Возврат каретки или любая функциональная клавиша позволяет пользователю либо «провалить»

```

Help Copy Delete Edit Print Modify Quit Rename Show cOmmand
C:\
  filename ext      size      date      time      attribute
  ANKETA      1965      01-20-87  12:29:30  [B       |
  ANKETA2     2320      01-20-87  12:24:50  [B       |
  BASIC       <DIR>      10-26-87  12:31:48  |       |
  BOOK       <DIR>      10-14-87  16:43:38  |       |
  FIG        <DIR>      01-19-87  17:00:26  |       |
  IAT        <DIR>      12-09-87  11:52:22  |       |
  ND         <DIR>      01-15-87  16:49:42  |       |
  KT         <DIR>      10-27-87  10:56:02  |       |
  STATG      <DIR>      12-15-87  11:09:24  |       |
  TEXT       <DIR>      10-13-87  12:44:06  |       |
  AUTOEXEC   BAK      199      10-13-87  12:48:06  [B       |
  CONFIG     BAK      56       10-16-87  13:11:58  [B       |
  AUTOEXEC   BAT      220      12-09-87  14:10:42  [B       |
  PRINT29    BAT      212      12-04-87  14:37:50  [B       |
  PRINT31    BAT      233      05-29-87  17:39:14  [B       |
  APPEND     COM      1725     07-07-86  12:00:00  [B       |
  ASCR       COM      3763     02-04-87  17:06:06  [B       |
  ASSIGN     COM      1523     07-07-86  12:00:00  [B       |
  CAT        COM      15872    01-13-87  17:08:00  [B       |
  COMMAND    COM      23612    07-07-86  12:00:00  [B       |
  DOS-EDIT   COM      2112     12-08-83  07:27:56  [B       |
F2- F3-top F4-bot F5-reread F6-sort F7-swap F8-in/exclude F9-back F10-mult

```

Рис. 3.4. Экран при работе с файловым монитором CAT

ся» в выбранный файл, чтобы просмотреть его текст или отредактировать его, либо выполнить данный файл, если он содержит выполнимые коды, либо перейти в новый каталог, если курсор указывает на один из каталогов, в которые можно перейти из текущего. Точно таким же способом можно вернуться назад по дереву каталогов.

Файловый монитор обычно содержит несколько командных строк. На рис. 3.4 показана система, имеющая две командные строки: внизу приведен список функциональных клавиш F1 — F10 и кратко указаны соответствующие им функции. Эти функции состоят в движении по текущему каталогу, перелистывании страниц (F1, F2), установке курсора на первый и последний файлы каталога (F3, F4), команде повторного считывания каталога (F5), используемой при замене гибкого диска. Полезна также команда сортировки файлов в каталоге, реализованная с помощью клавиши F6. Сортировка позволяет просмотреть каталог, расположив файлы по алфавиту, отсортировав их по расширению (при этом текстовые и командные файлы будут представлены в различных участках списка), по размеру, по дате создания. Нажав клавишу F7, пользователь может прочесть данные о каталоге, расположенном на другом диске. Весьма частой задачей при работе с файлами является стирание, переименование или копирование сразу некоторой группы файлов. Это могут быть файлы, в которых содержатся программные модули, относящиеся к какой-то одной программной системе, файлы данных и т. п. Для выделения группы файлов в системе используется клавиша F8. Нажимая на эту клавишу, можно пометить файл, на который указывает курсор, при этом изменится цвет символов в этой строке или появится некоторый служебный знак, указывающий на факт выделения данного файла.

Для того чтобы сообщить системе, что она должна выполнить операцию с группой файлов, достаточно нажать функциональную клавишу F10. Верхняя командная строка показывает, какие именно функции необходимо выполнить с данным файлом: просмотреть его содержание, стереть, скопировать, изменить атрибуты, переименовать. Кроме этого, в верхней командной строке имеются указания на выход из надстройки (Quit) и на приказ о выполнении любой из команд ДОС. При запросе на исполнение команд ОС эта команда может быть набрана в верхнем поле системы в специально отведенной для этого строке.

Перечисление основных функций системы выглядит немного громоздким, но в данном случае это впечатление кажущееся и не должно пугать читателя. Дело в том, что при работе с ПЭВМ в описанной нами системе максимально используются естественные для человека правила работы с пространственными структурами данных: движение курсора по экрану, пере-

листвование страниц и т. д. Оценить реальное удобство такой системы и все преимущества, которые она предоставляет пользователю, можно лишь проработав с ней некоторое время. Исходя из чисто практического опыта, можно утверждать, что описанная структура или любая структура, идеологически близкая к описанной, является тем разумным компромиссом, который делает работу с надстройкой много удобнее по сравнению как с прямой работой с ДОС, так и с работой с более изощренными системами класса WINDOWS.

Рассмотрим простой пример работы с такой системой. Предположим, что мы хотим просмотреть содержимое текущего каталога и скопировать несколько файлов на гибкий диск. Для этого необходимы следующие действия.

Набираем на клавиатуре команду CAT (если в данный момент надстройка не загружена) и видим на экране список файлов текущего каталога.

Клавишами PgUp и PgDn листаем каталог, как в обычном текстовом редакторе. Клавишами управления курсором выбираем те файлы, которые мы хотим скопировать, и функциональной клавишей F8 помечаем их специальным символом (звездочками).

Затем нажимаем клавишу F10 и клавишу C (копирование группы файлов). В верхней части экрана появляется вопрос: Куда копируем? Пишем в этой строке имя устройства A: и даем возврат каретки.

Теперь система копирует выбранную группу файлов, обозначая в каждый момент, какой файл копируется. Задача выполнена и мы снова оказываемся перед списком файлов текущего каталога.

Интересным примером организации надстройки над ДОС является система FDO: с ее помощью можно самому сформировать меню для выполнения тех или иных команд ДОС, задать управляющие конструкции, подгрузить к системе другие прикладные программы. Хотелось бы также упомянуть, что для ПЭВМ разработано несколько десятков удобных программ для выполнения одной или двух-трех распространенных операций, например, копирования и сравнения дисков (COPYIPC, COPYWRIT, DISKCOPY, BACKUP). В этих системах имеется удобное меню для быстрого и эффективного копирования и сравнения файлов.

Заканчивая параграф, можно сделать следующий вывод: любому пользователю ПЭВМ желательно иметь надстройку над ДОС типа файлового монитора. Начав работать с подобной системой, уже трудно себе представить, что когда-то можно было обойтись и без нее.

Мы уже встречались в § 1.2 с файлами, имеющими расширение BAT. Простейшая форма такого файла — просто последовательность команд ДОС, которые будут выполняться при вызове этого файла так, как будто мы набираем их на клавиатуре ПЭВМ.

Пакетные файлы прежде всего нужны для автоматического выполнения часто встречающихся операций, набирать которые на клавиатуре долго и утомительно. Например, при отладке программы приходится десятки раз набирать одну и ту же последовательность команд: вызов компилятора, сборку и запуск. Так, пользователь, отлаживающий свою программу TEST.C, для автоматического выполнения операций компиляции и сборки может написать в файл RUN.BAT следующий текст:

```
MSC/AL TEST;  
LINK TEST;  
TEST
```

Для запуска этой последовательности операций теперь достаточно набрать на клавиатуре RUN и дать возврат каретки.

Пакетные файлы могут использоваться и в более сложных режимах. Например, можно организовать подобие дополнительной команды ДОС, имеющей аргументы и выполняющей какую-либо полезную функцию. Так, «приходящие» пользователи ПЭВМ могут создать специальную команду, которая будет «подготавливать» ПЭВМ к работе с ними. Эта программа по введенной фамилии пользователя войдет в его каталог, загрузит текстовый редактор или прикладную программу, изменит параметры ДОС, словом, создаст привычную для него операционную обстановку. Другим важным пакетным файлом является файл AUTOEXEC.BAT, который автоматически выполняется после каждого включения или перезагрузки компьютера. Создание пакетных файлов по сути напоминает создание обычной программы: в ДОС существуют развитые средства для управления работой пакетных файлов. Эти средства мы и рассмотрим.

Начнем с примера, приведенного в начале параграфа. Основной недостаток такого пакетного файла состоит в том, что для компиляции файла с именем, отличающимся от TEST.C, придется написать другой пакетный файл. Можно ли этого избежать?

Пакетные файлы могут работать с параметрами. Смысл этих параметров точно такой же, как и при выполнении команд ДОС. Например, для выполнения произвольного файла с текстом программы на языке Си, разумно было бы дать команду «RUN имя файла», например

```
RUN TEST
```

Для того чтобы пакетный файл «узнал» о передаваемых в него параметрах, в ДОС принято следующее соглашение: параметры в тексте пакетного файла обозначаются знаком процента и номером. Для приведенного нами примера

```
MSC/AL %1;
```

```
LINK %1;
```

```
%1
```

Всего в пакетный файл можно передать девять параметров. Пусть, например, для выполнения программы необходимо откомпилировать и связать два файла. Эту операцию можно выполнить с помощью следующего пакетного файла (пусть он называется TWO.BAT):

```
MSC/AL %1;
```

```
MSC/AL %2;
```

```
LINK %1+%2;
```

```
%1
```

Для вызова такого пакетного файла теперь необходимо дать команду типа

```
TWO TEST1 TEST2
```

Пакетные файлы могут выполнять множество функций, которые при желании можно «скрыть» от пользователя, т. е. можно не писать текст выполняемых команд на экране. Для этого в пакетном файле можно дать команду

```
ECHO OFF
```

запрещает вывод команд пакетного файла на экран,

```
ECHO ON
```

разрешает вывод команд.

Заметим, что отображаться на экране не будут только сами имена команд пакетного файла. Например, если дать команду ECHO OFF и включить в пакетный файл команду DIR, то на экране будет показан список файлов текущего каталога, но не будет написано слово DIR.

Если вывод команд на экран разрешен, может возникнуть необходимость написать какое-нибудь сообщение для пользователя. Например, если в пакетном файле имеется команда копирования файла на гибкий диск, желательно иметь возможность написать на экран «Вставьте диск в дисковод». Для этого используется команда REM. Например, следующая строка в пакетном файле напишет сообщение на экран:

```
REM Вставьте диск в дисковод!
```

Но как быть, если пользователь еще не вставил дискету? Естественно, ему необходимо дать возможность подождать некоторое время и затем продолжить работу пакетного файла. Для этого служит функция

```
PAUSE сообщение
```

Например,

```
PAUSE Вставьте диск в дисковод!
```

После выполнения паузы на экране появится надпись «Нажмите любую клавишу, когда будете готовы» (Strike any key when ready). Нажав на любую клавишу, пользователь возобновит выполнение команд пакетного файла. Если вывод на экран запрещен при помощи команды ECHO OFF, то можно выдавать пользователю сообщения следующим образом: написать в файле

ECHO сообщение

Рассмотрим простой пример. Создадим пакетный файл, который по окончании рабочего дня будет копировать все файлы с заданным расширением из текущего каталога на гибкий диск. Назовем его EVENING.BAT. Вот как он может выглядеть:

PAUSE Вставьте дискету, на которую будем копировать файлы

ECHO OFF

COPY *.%1 A:

ECHO ON

REM Скопировал все, что нашел

Для копирования всех текстов с помощью этого пакетного файла достаточно дать команду EVENING TXT. После этого на экране появится надпись «Вставьте дискету, на которую будем копировать файлы», «Нажмите на любую клавишу, когда будете готовы». Вставив дискету, нажмем на любую клавишу и все файлы с расширением TXT будут скопированы на гибкий диск, при этом имена этих файлов будут показаны на экране. По окончании операции копирования на экране появится надпись «Скопировал все, что нашел».

А как быть, если пакетный файл не может выполнить команду, например, если он не нашел на диске текстовых файлов? Для этого в пакетном файле используются операторы безусловного перехода GOTO и условного перехода IF.

Оператор GOTO записывается следующим образом:

GOTO метка

Метка в пакетном файле обозначается двоеточием и именем. Например,

GOTO nofile

: nofile

gem Не нашел таких файлов

Оператор IF может иметь один из следующих видов:

IF [NOT] ERRORLEVEL число команда

IF [NOT] символ1 == символ2 команда

IF [NOT] EXIST имя файла команда

Параметр NOT в скобках не обязателен. Он может использоваться для отрицания логического условия.

Рассмотрим смысл этих команд. В первой форме записи оператора IF используется специальная системная переменная ERRORLEVEL. В большинстве случаев эта переменная равна нулю, если выполнение предыдущей команды или программы

прошло нормально и единице, если при этом произошли какие-либо ошибки. Заметим, что переход в операторе IF происходит тогда, когда переменная ERRORLEVEL равна или больше заданного числа. Например, результат проверки правильности компиляции может выглядеть следующим образом:

```
MSC/AL %1;  
IF ERRORLEVEL 1 ECHO Обнаружены ошибки!  
что эквивалентно условию ERRORLEVEL не ноль.
```

Вторая форма записи оператора IF предназначена для сравнения двух символьных переменных, например

```
IF %1 == %2 GOTO ABORT
```

```
:ABORT
```

REM Вы задали программе два одинаковых аргумента!

Третья форма записи оператора IF предназначена для поиска файлов с заданными именами. Например, следующий фрагмент программы выдает предупреждение и прекращает работу, если файл не записан на диске:

```
IF NOT EXIST %1 GOTO EXIT
```

```
:EXIT
```

REM Нет такого файла, прекращаю работу

Теперь мы узнали практически все, что необходимо для того, чтобы составлять сложные и большие пакетные файлы. Например, наш файл, компилирующий текст программы, может быть записан следующим образом:

```
:AGAIN
```

```
MSC/AL %1;
```

```
IF ERRORLEVEL 1 GOTO MISTAKE
```

```
GOTO LINK
```

```
:MISTAKE
```

PAUSE Обнаружены ошибки, входим в редактор текста

```
LEXICON %1. C
```

```
GOTO AGAIN
```

```
:LINK
```

REM Компиляция прошла успешно, начинаем сборку

```
LINK %1;
```

```
IF ERRORLEVEL 1 GOTO LERR
```

REM Сборка прошла успешно, выполняем программу!

```
%1
```

```
:LERR
```

REM Сборка не была выполнена, проверьте программу!

Этот пакетный файл будет компилировать текст программы. Если в процессе компиляции будут обнаружены ошибки, он автоматически вернется в текстовый редактор LEXICON, выйдя из которого пользователь снова начнет компиляцию и так до тех

пор, пока текст программы не станет правильным. Работа с таким пакетным файлом может оказаться очень удобной при разработке и отладке программы: обнаружение ошибок компиляции автоматически приведет в режим редактирования текста, что избавляет пользователя от необходимости набирать одни и те же команды на клавиатуре.

Приведем еще один пример пакетного файла. Пусть на ПЭВМ время от времени работают три пользователя: Иванов, Петров, Сидоров. Сидоров каждый раз приносит на дискете свои текстовые файлы и неумолимо заполняет ими диск компьютера, после чего немедленно входит в текстовый редактор. Петров начинает работу на ПЭВМ с разминки — играет в шахматы. Иванов наименее требователен: ему достаточно лишь войти в свой каталог и показать список файлов, используя файловый монитор CAT. Один из пользователей такой ПЭВМ вполне мог создать следующий пакетный файл (пусть он называется ENTER.BAT):

```
echo off
IF %1 == ivanov GOTO IVAN
IF %1 == petrov GOTO PETR
IF %1 == sidorov GOTO SIDOR
echo Кто Вы такой? Я Вас не знаю
goto END
:IVAN
echo Привет, Иванов! Вхожу в твой каталог
cd\text\ivanov
cat
goto END
:PETR
echo Привет, Петров! Запускаю шахматы
cd\text\petrov\games
if NOT EXIST CHESS.EXE goto PETT
chess
goto END
:PETT
echo прости Петров, шахматы кто-то стер!
goto END
:SIDOR
cd\text\petrov\sidorov
:AGAIN
echo on
pause Сидоров, вставляй свою дискету
echo off
if NOT EXIST a: *.txt goto SIDER
copy a: *.txt
echo Я все скопировал, Сидоров, вхожу в редактор!
lexicon
goto END
```

:SIDER

echo Ты какой диск вставил, Сидоров?

echo Копируй свои файлы самостоятельно!

:END

Мы предоставляем читателю самому разобраться в том, что произойдет, если набрать на клавиатуре

ENTER ivanov

ENTER petrov

ENTER sidorov

ENTER somebody

3.4. КАК ЗАДАТЬ КОНФИГУРАЦИЮ ПЭВМ

Как мы уже знаем, ДОС имеет множество удобных команд, позволяющих организовать на дисках иерархическую файловую структуру, задать тип приглашения ОС, выбрать режим работы с экраном, объявить маршрут движения по дереву каталогов и т. д. Кроме того, в ДОС имеются специальные системные команды для ускорения работы компиляторов, для объявления части оперативной памяти так называемым «виртуальным диском» и т. д. Все эти команды можно записать в специальные системные файлы ДОС — CONFIG.SYS и AUTOEXEC.BAT.

Каждый владелец ПК должен представлять себе процесс создания его конфигурации. В этом параграфе мы постараемся дать ряд практических рекомендаций по организации файловой структуры на диске и опишем процесс конструирования системных файлов ДОС.

Начнем с создания файловой структуры. Если в компьютере установлен фиксированный диск, то пользователю необходимо тщательно продумать, какие именно каталоги он создаст и какие программы будет в них хранить. Приведем несколько рекомендаций по удобной организации каталогов файлов ПЭВМ.

В корневом каталоге имеет смысл хранить минимальное число необходимых файлов, прежде всего командный процессор COMMAND.COM и системные файлы ДОС. В корневом каталоге также удобно разместить небольшие общесистемные программы и некоторые надстройки, например линкер, файловые мониторы или удобные программы для тестирования и копирования дисков. Все команды и файлы ДОС имеет смысл разместить в специально созданном каталоге ДОС. Заметим, что в корневом каталоге обязательно должны иметься файлы AUTOEXEC.BAT и CONFIG.SYS.

Для каждого языка, используемого на ПЭВМ, необходимо создать свой каталог, в котором будут храниться файлы компиляторов, интерпретаторов и все вспомогательные файлы. Если используются различные версии этого языка, то их желательно «разнести» между собой, создав каталоги более низкого уровня. Например, если используются две версии языка Паскаль — Турбо-Паскаль и Микрософт Паскаль — то желательно сделать два независимых каталога, например:

PASCAL\TURBO

PASCAL\MSOFT

Заметим, что для некоторых языков, например Си, структура каталога может быть довольно сложной: внутри него придется создать дополнительный каталог более низкого уровня, в котором будут расположены файлы, содержащие описания стандартных процедур языка Си.

Библиотеки различных языков (см. § 4.6) имеет смысл объединить в специальном каталоге LIB. Ниже мы рассмотрим инструкцию, которая «объясняет» линкеру, где именно искать библиотеки.

Иногда все компиляторы и интерпретаторы различных языков объединяют как каталоги более низкого уровня в специальном каталоге LANG. Нам представляется, что такое «сверхиерархическое» построение файловой структуры не всегда оправдано: достаточно, если каталоги двух-трех используемых языков программирования будут выходить непосредственно из корневого каталога. Другое дело, если в компьютере используется большое число различных языков программирования, но такая ситуация встречается очень редко.

В отдельных каталогах имеет смысл поместить и различные пакеты прикладных программ. Например, для БД придется завести каталог DBASE, для системы автоматизированного проектирования свой каталог CAD и т. п. Опять-таки, если используемое число стандартных пакетов программ относительно невелико (пять-шесть), соответствующие каталоги для их хранения имеет смысл вывести сразу из корневого каталога, если же таких пакетов очень много, то все их можно описать как каталоги, выходящие из некоторого «центрального» каталога прикладных пакетов программ.

Вряд ли имеет смысл создавать отдельный каталог для текстов, особенно если на ЭВМ работает несколько пользователей. Более логичной является такая структура: для каждого пользователя создать свой каталог, внутри которого будут созданы каталоги для текстов, программ, рисунков и т. п.

Вообще при создании файловой структуры на диске можно придерживаться следующего правила: стараться уменьшать степень ветвления при переходе к более низким уровням дерева каталогов, разумно ограничив «глубину» дерева и количество каталогов, выходящих из корневого. Это означает, что наибольшее разумное число каталогов, должно выходить из корневого, а число выходов из каталогов более низкого уровня надо стараться ограничить. Разумное число каталогов, выходящих из корневого, составляет десять-пятнадцать (при этом они еще одновременно видны на экране.ПЭВМ), при большем уровне ветвления его надо стараться ограничить. Из каждого каталога более низкого уровня обычно выходит пять-шесть каталогов. Разумная «глубина» дерева каталогов не должна превышать четырех-пяти уровней, иначе файловая структура станет крайне запутанной.

Итак, на наш взгляд, разумная организация каталогов на ПЭВМ общего назначения может выглядеть следующим образом

первый уровень: корневой каталог содержит файлы COMMAND.COM, AUTOEXEC.BAT, CONFIG.SYS; может содержать две-три системных программы, например линкер и файловый монитор;

второй уровень выходит из корневого каталога: десять-пятнадцать каталогов для команд ДОС, библиотек, компиляторов, прикладных программ и файлов пользователей;

третий уровень: каталоги содержащие различные версии языков, прикладных пакетов, редакторов, файлы пользователей (отдельные каталоги для текстов, программ, рисунков);

четвертый уровень: дополнительные каталоги (если необходимы), например, тексты пользователей (статьи, отчеты, доклады и т. п.).

Создание файловой структуры — только часть необходимой работы. При загрузке операционной системы ей необходимо сообщить сведения о существующем программном окружении, например об использовании специальных драйверов внешних устройств, о том, где хранятся библиотеки языков программирования, какие ветви каталога сделать прозрачными и т. д. Итак, предположим, что файловая структура продумана и частично создана. Каким образом сообщить компьютеру сведения о системных параметрах?

При загрузке ДОС система всегда начинает поиск в корневом каталоге файла CONFIG.SYS, в котором записываются директивы, предписывающие ДОС использовать те или иные системные функции. Если этот файл найден, система считывает из него необходимые сведения. Затем ищется файл AUTOEXEC.BAT, в котором содержится набор команд операционной системы, выполняемых при каждой перезагрузке компьютера, и, если этот файл найден, выполняются содержащиеся в нем команды. Какие сведения можно включить в файл CONFIG.SYS? Во-первых, системе можно указать, что исполняемые

команды можно прерывать одновременным нажатием клавиш Ctrl и C. Для этого достаточно написать команды

break-on (разрешает прерывание команд с помощью Ctrl-C)

break-off (выключает функцию прерывания команд)

При загрузке ДОС в памяти отводятся специальные буферы, в которые временно помещаются данные при выполнении операций записи и чтения с диска. По умолчанию число таких буферов равно двум, однако их можно задать и больше (до 255). Увеличение числа буферов ускоряет операции при работе с различными пакетами программ, например при компиляции и сборке. Оптимальным числом буферов можно считать 10—20 (большее их число занимает слишком большой объем оперативной памяти). Для установки необходимого числа буферов используется команда buffers-число, например

buffers-20

ДОС позволяет записывать время и дату в различных форматах. Например, в американском стандарте дата записывается как месяц:число:год, а в европейском — как число:месяц:год. Способ задания даты можно изменить, указав в файле CONFIG.SYS

country=код

код 001 соответствует американскому стандарту, код 044 — европейскому.

При загрузке ДОС можно установить драйвер клавиатуры ANSI.SYS и драйвер виртуального диска RAMDRIVE.SYS. Первый из них позволяет управлять отображением символов на экране: позицией курсора, цветом символов. Второй позволяет использовать часть оперативной памяти под виртуальный диск. С этим диском можно обращаться точно так же, как и с обычным, но работа с ним происходит гораздо быстрее. Это очень удобно при выполнении ряда операций, требующих частого обращения к дискам, например при компиляции программ. Для задания типа используемого драйвера в файле CONFIG.SYS можно дать команду

device=ansi.sys

или

device=ramdrive.sys размер сектор число входов

где «размер» — размер виртуального диска в килобайтах (минимум 16 Кбайт); «сектор» — размер сектора в байтах (128, 256, 512 или 1024); «число входов» — сколько каталогов можно создать на виртуальном диске (от 2 до 1024). Типичный размер виртуального диска на ПЭВМ равен 64 Кбайт.

Например, следующая строка создаст виртуальный диск емкостью 360 Кбайт (если, конечно, хватит оперативной памяти), размер сектора 128 Байт и даст возможность создать на виртуальном диске 64 каталога:

device=ramdrive 360 128 64

Виртуальному диску присваивается первая свободная буква в списке дисководов компьютера. Например, если в ПЭВМ установлен один фиксированный диск C, то виртуальный диск будет обозначаться буквой D. Заметим, что вся информация, содержащаяся в виртуальном диске, исчезает после выключения компьютера.

В ДОС можно задать возможное число одновременно открытых файлов. Это делается с помощью команды

files=число,

где число может принимать значения от 8 до 255. Обычно разрешается открывать 20 файлов одновременно.

Типичное содержание файла CONFIG.SYS:

break=on

country=044

files=20

buffers=20

device=ANSI.SYS

device=RAMDRIVE.SYS 64 128 32

Этот файл разрешает прерывать программы сочетанием клавиш Ctrl-C, устанавливает дату в европейском стандарте, открывает 20 промежуточных буферов, разрешает одновременно открывать до 20 файлов, использует драйвер ANSI.SYS и создает виртуальный диск емкостью 64 Кбайт.

После того как установлены параметры, описанные в файле CONFIG.SYS, система начинает выполнение пакетного файла AUTOEXEC.BAT. Какие команды имеет смысл включить в этот файл?

Во-первых, необходимо указать системе те каталоги, в которых по умолчанию будет проводиться поиск пакетных или выполнимых файлов с помощью команды PATH. Во-вторых, необходимо задать вид приглашения системы (показывать или нет текущий каталог пользователя). Необходимо также установить время и дату и, возможно, загрузить одну из используемых программ или перейти в какой-то из каталогов ПЭВМ. Если все библиотеки языка находятся в каталоге LIB, то необходимо дать специальную команду

```
SET LIB=C:\LIB
```

Для файлов расширения языка Си предусмотрена специальная команда

```
SET INCLUDE=имя каталога
```

Если в системе предусмотрено использование виртуального диска, то необходимо включить в файл следующую команду:

```
SET TMP=первое свободное логическое имя диска.
```

Например, если в системе установлены гибкий диск C и один фиксированный диск D, то необходимо дать команду

```
SET TMP=D:\
```

Эта команда сообщает ряду системных программ, в том числе линкеру и компиляторам, о существовании виртуального диска. При этом промежуточные файлы этих программ создаются именно на этом диске, что ускоряет их работу и уменьшает число обращений к фиксированному или гибкому диску.

Типичное содержание файла AUTOEXEC.BAT:

```
echo off
path=c:\;c:\cc;c:\text;c:\basic;
SET INCLUDE=C:\CC\INC
SET LIB=C:\LIB
SET TMP=D:\
cd\text
prompt $p$g
cls
date
time
```

После каждого изменения описанных здесь файлов, пользователь обязательно должен загрузить ПЭВМ, тем самым «активировав» установленные параметры конфигурации.

3.5. РАБОТА С ТЕСТИРУЮЩИМИ ПРОГРАММАМИ ПЭВМ

По мере развития ПЭВМ и расширения числа периферийных устройств возросло и число различных пакетов прикладных программ для тестирования ПЭВМ. Число таких программ составляет несколько десятков и, надо сказать, что они значительно облегчают жизнь пользователей. По категории применения эти программы разделяются на две большие группы: программы тестирования электроники компьютера и программы тестирования гибких и фиксированных дисков.

Первая группа программ предназначена для проверки правильности функционирования различных составных частей ПЭВМ. Обычно эти программы поставляются в комплексе. Поскольку в настоящее время выпускается несколько десятков типов ПК класса IBM PC, отличающихся конструкцией, то и эти программы различаются между собой. Типичная структура таких программ (например, программа DIAGS для компьютера IBM PC XT) — набор меню для тестирования функций памяти, процессора, экрана и клавиатуры.

Меню программ тестирования памяти позволяет проверить исправность микросхем оперативной памяти: при их вызове осуществляются операции записи и считывания в память. Если при этой операции будут обнаружены ошибки, то программа выдаст сообщение об адресе повреждений и в некоторых случаях о расположении микросхемы, которую необходимо заменить. В наиболее изощренных пакетах тестирования при этом на экране возникает условное изображение расположения микросхем на плате процессора и красным цветом помечается неисправная схема.

Программы тестирования процессора (и в общем случае всей платы процессора) позволяют локализовать некоторые типичные неисправности, заставляя процессор выполнять набор тестовых операций. Чаще всего с помощью этих программ определяется правильность установки различных перемычек на системной плате ПЭВМ. В различных моделях компьютеров определенные сочетания таких перемычек сообщают компьютеру сведения о конфигурации системы. Тестирующая программа считывает установленную конфигурацию и выдает ее пользователю в естественном виде, например:

В вашем компьютере установлен

1 фиксированный диск

2 флоппи-диска

монохромный монитор

математический сопроцессор

Если эта конфигурация неправильна, то выберите соответствующую позицию.

Теперь пользователь с помощью клавиш управления курсором выбирает те разделы конфигурации, которые не соответствуют истинным. Например, если в системе в действительности установлен только один флоппи-диск, то выбрав данную строчку экрана и нажав на возврат каретки, пользователь видит на экране примерно следующее сообщение:

Если в вашей системе установлен 1 гибкий диск, откройте компьютер и переставьте перемычку 3 в положение: контакты 1, 2 замкнуты, 2, 3 — разомкнуты.

Программы тестирования графической платы и дисплея позволяют генерировать тестовые изображения и подбирать регулировку дисплея, проверять функционирование графического контроллера. Наконец, программы тестирования клавиатуры позволяют найти неисправные клавиши.

Такие тестирующие программы используются довольно редко, как правило при установке ПЭВМ или в случае сбоев в ее работе. Гораздо более распространенной ситуацией является поломка дисков из-за неисправности дисководов, бросков питания, попадания грязи и т. п. Мы уже упоминали, что стандартными средствами ДОС произвести диагностику диска довольно трудно: стандартная команда CHKDSK только выдает количество неисправных кластеров на диске. В тех же случаях, когда необходимо локализовать повреждение, попытаться спасти данные, приходится пользоваться специальными тестирующими программами.

Мы уже упоминали, что в некоторых надстройках над ДОС используется ряд подобных функций. Например, с помощью системы PCTOOLS можно локализовать неисправные секторы, просмотреть коды, записанные в определенных секторах диска, изменить их и т. п. Более универсальными и удобными, однако, являются специальные системы для тестирования дисков.

Системы для тестирования дисков состоят из набора меню, предоставляющего пользователю широкий набор функций. Эти функции можно сгруппировать следующим образом:

- выбор объекта исследования,
- получение информации о выбранном объекте,
- восстановление сохранившейся информации.

В качестве объекта исследования можно выбрать собственно диск, набор его секторов или один сектор, файл, стертый ранее файл, каталог файлов.

В разделе получения информации об объекте можно получить техническую информацию о диске (сколько сторон, секторов, какой объем занят, где расположены поврежденные участки и т. п.). Существует также возможность построить в графическом режиме карту распределения памяти на диске. При этом, как показано на рис. 3.5, весь экран представляет условное распределение дискового пространства, различными цветами отображаются занятые участки, файлы пользователей, системные файлы, поврежденные сектора. Теперь можно в таком же графическом режиме посмотреть, где на диске расположен выбранный ранее файл или сектор, найти его адрес. Достаточно установить курсор на выбранный участок памяти диска и, нажав на возврат каретки, вывести на экране содержимое памяти диска в текстовом или двоичном формате. Как и в текстовом редакторе, можно изменять содержание записанной информации. Существует также возможность задать системе любое сочетание символов или кодов и просканировать диск с целью локализации области памяти, в которой записано это сочетание. Выбранную область памяти на диске можно скопировать на другой диск.

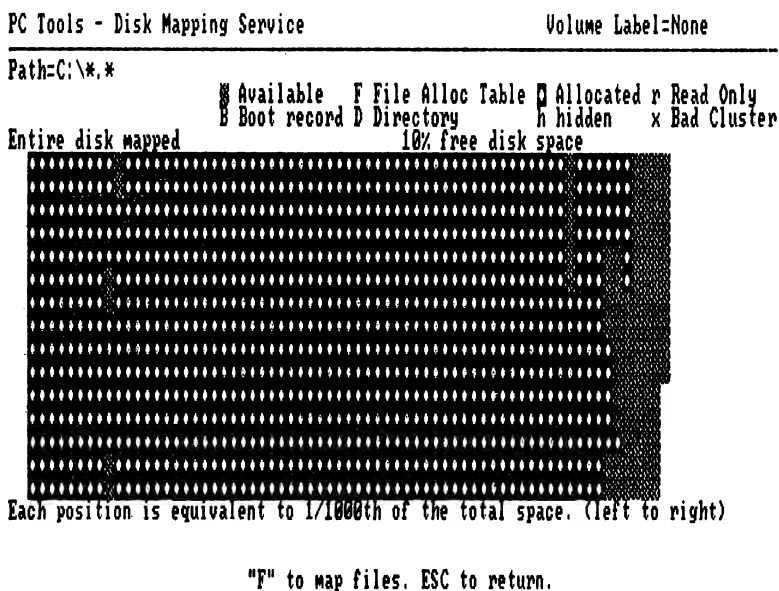


Рис. 3.5. Экран в режиме получения информации о состоянии фиксированного диска

В разделе восстановления информации можно попытаться «обойти» испорченные участки диска: при этом будет сделана попытка восстановить файловую структуру, каталоги, переписать стертые или испорченные файлы на диске таким образом, чтобы устранить из них испорченные участки. Такая процедура, конечно, не всегда может привести к успеху.

Рассмотрим несколько типичных ситуаций, в которых использование системы такого рода неоправдано.

Ситуация 1. Фиксированный или гибкий диск неожиданно сломался: система обращается к нему, но не находит файлов. По-видимому, на диске каким-то образом испортилась область памяти, в которой содержится системная информация о файловой структуре. Сделаем попытку восстановить ее. Попытка заканчивается неудачей. На диске записан плод длительного труда: текст или программа на Фортране, начинающаяся с сочетания PROGRAM SPP1. Что еще можно предпринять?

Попытаем счастья: предположим, что область диска, в которой хранится текст программы, не испорчена или испорчена частично. Попросим у системы просканировать весь диск и найти то место, в котором записано сочетание символов PPROGRAM SPP1. Для этого войдем в раздел меню поиска. Система задаст нам вопросы «Что искать» (вводим наше сочетание символов) и «Где искать» (задаем поиск на всем диске). Теперь система начинает сканирование дискового пространства. Через некоторое время она находит это сочетание и пишет: «Строка найдена, адрес:...». Теперь нам достаточно дать возврат каретки и увидеть текст нашей программы. Листая страницы, как в текстовом редакторе, мы видим следующие участки диска и убеждаемся, что весь текст или какая-то его часть не испорчены. Теперь мы замечаем начальный и конечные адреса на диске, в которых записана наша программа, и вызываем функцию копирования участка памяти, в которой указываем эти адреса и имя нового файла на гибком диске, в который будет записана эта информация.

Ситуация 2. При работе с диском вдруг обнаруживается, что почему-то испорчены некоторые секторы. Предположим, что это произошло не из-за механических поломок, а вследствие сбоя контроллера или броска напряжения питания, и попытаемся исправить эту ситуацию.

Вызовем тестирующую систему и попросим определить адреса испорченных участков дисковой памяти. Если такой адрес найден, войдем в режим просмотра двоичной информации, записанной на диске. Предположим, что нашим глазам предстает следующая ситуация: предыдущий и последующий секторы диска заполнены определенными кодами (например, нулями), но в испорченном секторе почему-то часть памяти заполнена нулями, а в нескольких адресах записаны какие-то случайные числа. Попытаемся избавиться от них и дадим команду на заполнение этого участка от дисковой памяти нулями. После этого снова запустим тест. Ошибки исчезли! Авторам приходилось несколько раз сталкиваться с аналогичными ситуациями и в ряде случаев удавалось спасти свою информацию или избавиться от ошибок на диске таким способом.

Научиться работать с подобными системами очень легко, так как их структура максимально приспособлена к использованию неподготовленным человеком: на экране всегда имеются подсказки, объясняющие, какую именно функцию выполняет та или иная программа. Наибольшую известность среди систем тестирования дисков получили программы Norton Utilities и DiskLook (автор — P. Norton).

Глава 4. ПРОГРАММИРОВАНИЕ БЕЗ ПЕРСОНАЛЬНОЙ ЭВМ

Как Вы, надеемся, убедились, научиться управлять персональным компьютером можно гораздо быстрее, чем научиться управлять автомобилем. Причина заключается в том, что на самом деле мы всякий раз работали не непосредственно с компьютером, а с устройством «компьютер плюс прикладная программа». Это устройство (пишущая машинка, записная книжка, графопостроитель и т. д.) могло быть простым в обращении или не очень, могло служить то одной цели, то совершенно другой, или совмещать в себе множество разных возможностей — все зависело от прикладной программы, которую мы предварительно вводили в компьютер. Именно разнообразием пакетов прикладных программ и объясняется популярность ПК среди людей совершенно разных занятий, порой вовсе не имеющих отношения к программированию.

Не секрет, что для создания всех пакетов опять-таки использовался ПК — это еще одна область его применения. Как же создаются прикладные программы? В этой главе мы попытаемся частично ответить на этот вопрос. Не исключено, что Вы сами сможете впоследствии написать какую-нибудь программу, которая окажется Вам полезной. Однако предупреждаем — не все в этой главе будет так просто, как было раньше. Труд программиста, как и труд любого профессионала, требует определенной подготовки. Не стоит рассчитывать и на то, что Вам удастся написать прикладной пакет вроде тех, о которых шла речь — даже для профессионала это огромный объем работы. Правда, у Вас есть важное преимущество перед фирмой-изготовителем прикладных пакетов — Вы пишете программу персонально для себя и значит способны куда точнее подогнать ее к конкретной задаче, избежав ненужной общности. Не исключено также, что вопрос «Как создают программы?» просто порожден Вашей любознательностью, как и вопрос «Из чего делают порох?» или «Почему бывает гроза?» И в этом случае стоит взяться за эту главу. Ну а если Ваше любопытство не распространяется на программирование, и набор уже имеющихся прикладных программ для ПК Вас полностью удовлетворяет, оставшиеся главы можно и пропустить.

Программирование ЭВМ как искусство заставить машину выполнить желаемые действия появилось, видимо, одновременно с появлением программируемых машин, т. е. где-то в 40-х годах. История напоминает нам, что были эти машины огромны, ненадежны и малопонятливы, так что ремесло программиста более всего походило на профессию заклинателя змей. Со временем программирование упростилось, что было связано как с эволюцией ЭВМ (эволюционировало и «тело» — элементная база, и «разум» — программное обеспечение), так и с развитием

научных основ программирования. Однако профессия программиста и по сию пору сохраняет некоторый налет шаманства и мистики: по-прежнему мало кто может поручиться, что руководимый его программой компьютер не преподнесет какую-нибудь неожиданность. И, как показывает печальная практика, не зря — ряд впечатляющих примеров компьютерных ошибок и их последствий можно найти, скажем, во введении книги [36]. Там же содержится и ряд предложений — как защитить себя от ошибок в программировании. Ряд книг, среди них в первую очередь [24, 25], посвящен общим методам составления программ, позволяющим обойтись без так называемой «отладки» (так на программистском жаргоне называется небезызвестный «метод тыка»).

Все сказанное не означает, что если Вы новичок в программировании, то следует прежде засесть за перечисленные только что книги — на первых порах, при написании простых программ, Вам вполне хватит ПК и этой книжки. Как правило, необходимость систематического подхода начинает по-настоящему ощущаться лишь после многочисленных «поединков» с компьютером, из которых не всегда удается сразу выйти победителем.

4.1. КЕМ КОМАНДУЕТ ПРОГРАММА

Представим себе, что компьютер выполняет программу... Что, кстати, при этом нужно представлять? Видимо, разные люди представляют разное. Радиотехник, быть может, подумает об электрических токах, которые текут в различных схемах, моделируя логические операции. Знаток архитектуры ЭВМ расскажет, как центральный процессор выбирает из памяти команды и что происходит при их выполнении; специалист по операционным системам объяснит, как программа грузится с диска, как ей передается власть над машиной, и чем эта власть ограничена. Мы же, пользователи, видим, что машина подает характерные «признаки жизни»: на видеомониторе появляются какие-то надписи, из динамика временами раздаются звуки, «оживают» связанные с компьютером внешние устройства: что-то печатает принтер, что-то рисует графопостроитель, приходит в движение связанный с компьютером через цифро-аналоговый преобразователь исполнительный механизм и т. п.

Все эти действия совершаются под командованием находящейся сейчас в ПК прикладной программы. С точки зрения программы, все перечисленные только что устройства являются *устройствами вывода информации* — с их помощью информация, выработанная прикладной программой, попадает из компьютера во внешний мир.

Редко, однако, бывает так, что поведение компьютера жестко определяется программой. Практически все описанные нами прикладные программы для ПК — диалоговые, их поведение зависит еще и от того, какие клавиши на клавиатуре мы нажимаем во время работы программы. Клавиатура — это одно из устройств для ввода информации в компьютер. Помимо опроса клавиатуры, программа, работающая в ПК, может пожелать прочесть что-нибудь с диска или измерить напряжение на входе аналого-цифрового преобразователя и т. п. Все это позволяет программе узнавать что-то новое о внешнем мире по мере своей работы, а пользователю — влиять на работу программы.

Итак, *устройства ввода* обеспечивают передачу информации программе от того или иного внешнего источника. При этом у устройства ввода обычно двойное подчинение: пользователю (или еще какому-нибудь источнику информации) и программе. Происходит это так: извне устройство получает команды вроде «передай программе это». Программа же шлет устройству ввода примерно такие команды: «сообщи, не было ли чего для меня», «ах, раз было, расскажи, что именно», «а вот эти сообщения выброси, они мне не нужны» и т. д. Таким образом, устройства ввода тоже выполняют приказы программы.

Основной задачей прикладной программы, однако, является не ввод и вывод информации, а ее хранение и обработка. Так, ясно, что программа, строящая на мониторе проекции трехмерного объекта, должна как-то хранить в памяти этот объект и уметь вычислять его проекцию. При этом способ хранения информации об объекте будет сильно влиять на удобство и время построения проекции. Различные структуры для хранения информации — структуры данных — это еще один тип объектов, которыми командует программа.

Итак, наблюдая за работающим компьютером, мы установили, что программа должна уметь управлять следующими устройствами:

- вывода информации (например, видеомонитором),
- ввода информации (например, клавиатурой),
- хранения и обработки информации (оперативной памятью и процессором).

Любая программа, на каком бы языке она не записывалась, по сути дела задает последовательность команд, обращенных к этим устройствам. Поэтому давайте сначала разберемся, как выглядит каждое из этих устройств с точки зрения программы, т. е. какие команды они могут выполнять и что при этом с ними происходит.

Прежде чем начать рассказывать про команды и тем более составлять из них программы, придется условиться о каком-то языке для записи команд и программ. Можно было бы сразу записать все на одном из существующих языков программиро-

вания для ПК, но нам не хотелось бы начинать с этого. Причин тому несколько: от стремления избежать ненужных на первых порах деталей, до нежелания использовать иностранный язык (увы, все известные нам языки программирования для ПК почему-то состоят из английских слов). Поэтому мы построим наше изложение так:

1) расскажем, какие бывают команды, и как из команд создаются программы (при этом будем использовать свою систему записи);

2) объясним, как полученные программы перевести на тот или иной язык для ПК и выполнить.

Поверьте, что перевод программы — это чисто рутинная работа, куда более простая, чем работа переводчика с русского на английский. А вот на первом этапе многое придется понять, причем безо всякого компьютера — он нам не поможет.

Итак, начинаем рассказ про команды работы с устройствами.

4.2. УСТРОЙСТВО ВЫВОДА: ВИДЕОМОНИТОР

Снаружи он выглядит просто — это ящик с телевизионной трубкой. На нем есть несколько ручек с типично телевизионным назначением — яркость, размер по вертикали и т. п. Они позволяют сделать картинку на экране чуть шире или уже, ярче или бледнее, но на содержание самой картинки не влияют никак — вывод картинки на экран является монополией программы. Что же программа может изображать на мониторе?

Будем для простоты считать, что вывод на монитор происходит в том режиме, который устанавливается сразу после включения машины (позже мы расскажем и об остальных режимах). В этом случае на экране можно изображать только символы. Размеры символов, места, в которых они могут изображаться, и сам набор изображаемых символов жестко заданы: экран поделен на 25 строк, в каждой строке может изображаться по 80 символов, т. е. всего имеется $25 \times 80 = 2000$ мест для изображения символов. Их называют позициями или знакоместами. Вид символов может несколько различаться от машины к машине. Будем считать, что среди символов присутствуют большие и маленькие латинские буквы (что всегда так и есть), большие и маленькие русские буквы (это бывает очень часто, хотя и не всегда), знаки препинания и некоторые другие символы, удобные для формирования картинок (так называемые символы псевдографики, они есть на любой машине, но на разных машинах могут выглядеть по-разному). Не исключено, что Ваш монитор цветной — тогда на нем можно изображать символы нескольких цветов, но давайте пока не обращать внимания на цвет. Вспомним еще о том, что на экране имеется курсор. Он может находиться в любой из 2000 позиций.

Для того чтобы программа могла полностью управлять содержимым экрана, она должна уметь изображать любой символ в любой позиции и перемещать курсор в любую позицию. Поэтому кажется естественным такой набор команд:

изобразить такой-то символ в такой-то позиции экрана;
поместить курсор в такую-то позицию экрана.

Как видим, команды просты и позволяют придать экрану любой вид. Однако уже здесь начинаются парадоксы: в языках программирования не предусмотрено стандартных команд вывода на монитор, подобных придуманным нами. Можно найти объективные причины этого, но факт остается фактом: общепринятым является другой способ вывода информации на монитор, и мы о нем сейчас расскажем.

Стандартной командой вывода на экран в большинстве языков является команда, которую мы обозначим так:

напечатать («любая строка»)

По этой команде соответствующая строка символов выводится на экран. Строка выводится слева направо, символ за символом. При этом первый символ строки выводится в ту позицию, в которой в данный момент находится курсор, после чего курсор сдвигается на одну позицию вправо. Затем в эту позицию выводится следующий символ и т. д., пока не выведется вся строка.

Пример. Если фрагмент экрана выглядит так, как показано на рис. 4.1, а, то после выполнения команды «напечатать («Пробная строка»)» он примет вид, показанный на рис. 4.1, б.

Если команда «напечатать» выполняется несколько раз или печатаемая строка достаточно длинная, то курсор дойдет до самой правой позиции экрана. Тогда при необходимости печатать дальше он перейдет в крайнюю левую позицию на следующей строке и печать продолжится

Пример. Если в ситуации, иллюстрируемой рис. 4.1, б, подать команду «напечатать («Еще одна строка»)», то получится картинка, изображенная на рис. 4.1, в.

Ясно, что по мере вывода строк на экран курсор наконец дойдет до правого нижнего угла экрана. Команда «напечатать» устроена так, что в этом случае при попытке вывести что-то еще

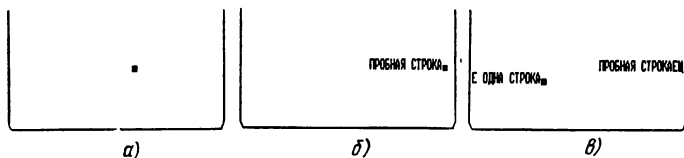


Рис. 4.1. Экран ПЭВМ при выводе текста:

а) исходное состояние; б) при добавлении текста; в) перевод строки

произойдет так называемое роллирование: весь текст на экране сдвинется на одну строку вверх. При этом строка, которая была верхней, исчезнет, а снизу появится пустая строка (т. е. строка, состоящая из пробелов). Курсор окажется в первой позиции этой пустой строки и текст будет выводиться туда. Именно так роллируется экран, например при диалоге с операционной системой — проверьте!

Есть еще одна команда, допускаемая в любом языке и позволяющая распределять выводимый текст по строкам. Мы запишем ее так:

перевести __ строку

Она сдвигает курсор в крайнюю левую позицию следующей строки (при необходимости роллируя экран), так что дальнейший текст, выводимый командой «напечатать», будет располагаться с самого начала строки.

Упражнение. Как будет выглядеть экран, если подать команду «перевести строку» 50 раз подряд?

Ответ. Независимо от начального состояния экрана он окажется пустым, а курсор попадет в левый нижний угол экрана.

Упражнение. Пусть экран вначале пуст (заполнен пробелами), курсор находится в левом верхнем углу экрана и выполняется такая последовательность команд:

напечатать («* * * *»)

перевести __ строку

напечатать («* *»)

перевести __ строку

напечатать («* *»)

перевести __ строку

напечатать («* *»)

Что в результате появится на экране?

Ответ. Будет нарисована буква «П».

Упражнение. Измените программу так, чтобы рисовалась буква «Н».

Не пренебрегайте этими упражнениями — возможно, это первые программы, которые Вы пишете. Отложим перевод этих программ на компьютерный сленг, но некоторые из них мы обязательно переведем. Давайте условимся: программам, которые мы планируем впоследствии выполнить на ПК, будем давать порядковые номера. Программа 1 пусть будет предельно простой:

ПРОГРАММА 1

напечатать («Программа 1 выполнена!»)

КОНЕЦ ПРОГРАММЫ

Словами ПРОГРАММА и КОНЕЦ ПРОГРАММЫ мы просто хотим придать тексту программы законченность (такое обрамление есть и в реальных языках программирования, правда, не во всех).

Видно, что команда «напечатать» и сложнее, и слабее тех команд, что вначале предлагали мы: она позволяет выводить текст лишь туда, где находится курсор, при этом самим курсором мы управлять практически не можем. На самом деле эта команда — наследие тех времен, когда все сообщения выводились на бумажную ленту (например, с помощью электрической пишущей машинки). Экран как раз и моделирует такую ленту, а операция роллирования — операцию продергивания этой ленты вверх при переходе на следующую строку.

Итак, при помощи команды «напечатать» мы способны вывести на экран любой текст. Однако мы умели делать это и с помощью команды `echo` в файле типа `.BAT`. Ясно, что программа, которая печатает на экране то, что заранее известно, практически бесполезна. Мы хотим, чтобы программа могла сообщить нам что-то новое, чего мы не знали, например, чему равно тысячное простое число или как выглядит график функции $f(x)$. Для этого программа должна выглядеть как-то так:

вычислить что-то

напечатать (то, что вычислено)

Пришла пора разобраться в том, как и где это что-то образуется, т. е. как программа хранит и обрабатывает информацию.

4.3. ПЕРЕМЕННЫЕ И КАК ИМИ УПРАВЛЯТЬ

Элементарный объект, которым умеет управлять компьютерная программа — это *переменная*. Неформальное определение переменной может быть, например, таким: это место для хранения информации. Каждая переменная имеет *имя*, позволяющее отличить ее от других переменных. Кроме того, каждая переменная, как хранитель информации, имеет свою определенную «специальность», т. е. она умеет хранить информацию лишь определенного сорта и определенного объема. Например, может оказаться, что переменной с именем `N` можно доверять для хранения лишь целые числа, да и то не больше одного сразу. Как выглядит команда «передачи на хранение»? Вот как: имя переменной := число

Справа от специального знака «:=» пишется то, что мы хотим сохранить, а слева — где (в какой переменной) мы хотим это сохранить. Например, команда «`N := 4`» поместит в переменную `N` число 4. Это число будет там лежать, пока не будет выполнена команда «`N := еще что-нибудь`». Программисты в таких случаях говорят, что переменной `N` *присвоено значение* 4, и переменная `N` будет *иметь значение* 4 до тех пор, пока ей еще чего-нибудь не присвоят. Саму команду называют *командой присваивания значения*. Справа от знака присваивания можно писать все, что, с нашей точки зрения, является числом — любое арифметическое выражение. Например, допустимы записи

$N := 3 + 1$, $N := 2 * 2$ и т. д.

Каждый конкретный язык программирования, конечно, накладывает свои ограничения на допустимые знаки операций и способ записи выражений. Так, знак умножения почти всегда обозначают звездочкой и т. д. Элементами выражений могут быть как числовые константы, так и те числа, которые в момент вычисления выражения хранятся в каких-либо переменных, т. е. можно писать команды вроде:

$N := 12 +$ то, что сейчас хранится в M
(будем считать, что переменная с именем M тоже предназначена для хранения целых значений). Это означает, что нужно сложить 12 и то число, которое сейчас в M , и результат поместить в N . Переменная M при этом, естественно, не меняется — команда присваивания меняет значение лишь той переменной, что написана слева от знака «:=». В языках программирования последнюю команду записывают короче: « $N := 12 + M$ ». Возможна и такая команда:

$N := N + 1$

означающая, что надо *сначала* взять число, лежащее в N , прибавить к нему 1, и лишь *затем* полученный результат поместить в N . Таким образом, то N , что справа, и то, что слева, имеют разный смысл.

Еще одно замечание. Переменная всегда хранит ровно одно значение — ни больше, ни меньше. Это означает, что при присваивании переменной нового значения старое бесследно теряется. Это также означает, что даже до того, как переменной хоть что-то присвоят, в ней хранится какое-то значение (как правило случайное, или не имеющее отношения к тому, что нам нужно).

Если каждая из каких-либо двух переменных умеет хранить все то, что умеет хранить другая, говорят, что эти две переменные *одного типа*. С переменными *целого* типа (умеющими хранить целое число) мы уже познакомились. Чуть позже мы расскажем и о других типах.

Введенная нами ранее команда «напечатать» позволяет выводить на экран монитора также значения переменных. Это можно записать так:

напечатать (это, и это, и это,...)

где внутри скобок можно перечислить через запятую как строки, заключенные в кавычки, так и арифметические выражения (вроде тех, что допускаются справа от знака «:=» в команде присваивания). Например, возможна такая команда:

напечатать («Путь равен», $V * T + 200$, «Y=», Y)

Исполняя эту команду, компьютер сначала напечатает текст «Путь равен» (без кавычек!), затем компьютер вычислит выражение $V * T + 200$ и напечатает результат, потом будет напечатан текст «Y=», и наконец, будет напечатано текущее значение переменной Y. Например, если V , T , Y — целые пере-

менные, текущие значения которых равны 15, 3 и 27 соответственно, то на экране появится надпись:

Путь равен 245 $Y=27$

А что произойдет с переменными V, T, Y в результате выполнения такой команды? Ровным счетом ничего — ведь команда эта управляет только экраном. Так что значения переменных V, T, Y остались прежними — мы только «посмотрели» на хранящиеся в них числа, чтобы вычислить арифметическое выражение или просто напечатать соответствующее число.

Упражнение. Пусть A и B — целые переменные, причем $A=2$, $B=3$ (будем в дальнейшем использовать такую краткую запись вместо громоздкого «текущее значение переменной A равно 2, текущее значение переменной B равно 3»). Что произойдет в результате выполнения такой последовательности команд:

$A := B; B := A$

Решение. После выполнения первой команды значение A станет равным текущему значению B, т. е. 3. Старое значение A потеряется. Поскольку A и B станут равными, вторая команда ничего не изменит.

Ответ. $A=B=3$.

Упражнение. Напишите программу, в результате выполнения которой переменные A и B обменяются значениями.

Ответ. Воспользовавшись дополнительной целой переменной X, получим такую программу:

$X := A; A := B; B := X;$

Построим по результату последнего упражнения такую программу:

ПРОГРАММА 2

A, B, X : цел

$A := 2; B := 3$

напечатать («До обмена $A=$ », A, « $B=$ », B)

$X := A; A := B; B := X$

перевести — строку

напечатать («После обмена $A=$ », A, « $B=$ », B)

КОНЕЦ ПРОГРАММЫ

Строчка «A,B,X : цел» сообщает, что программе потребуются 3 целых переменных, которым даются имена A, B и X. Похожие строки (называемые описаниями переменных) есть и в реальных программах — они информируют компьютер, например, о том, сколько места для хранения информации требуется отвести данной программе.

Переменные могут быть и других типов, не обязательно целые. Конкретный набор допустимых типов зависит от языка, но несколько типов переменных, встречающихся почти во всех языках особенно популярны.

Один из них — *вещественный* (мы будем обозначать его «вещ» в наших программах). Значениями переменных этого типа являются вещественные (они же действительные) числа. Конечно, вещественная переменная способна хранить лишь несколько значащих цифр числа, т. е. компьютер не способен в точности выполнить команду «сохранить в действительной переменной PI число π » (на это ему потребовалась бы память бесконечного объема). Запись вещественных чисел в языках программирования похожа на общепринятую, только вместо десятичной запятой используется точка, а знак «Е» в записи числа означает «умножить на десять в степени». Примеры записи вещественных чисел: 1.0, —0.0000153, —21.1E—3, 1E6 (два последних числа означают —0.0211 и 1000000).

Переменные *логического* («лог») типа умеют хранить лишь два различных значения — ИСТИНА и ЛОЖЬ. Позже мы обязательно встретимся с переменными этого типа.

Любая переменная *символьного* («симв») типа умеет хранить один символ. Например, если С — символьная переменная, то допустимы команды вроде:

С := 'А' С := ' ' (пробел) и т. д.

Значение символьной переменной может быть выведено на экран при помощи команды «напечатать». Важную роль играет символьная переменная в команде ввода с клавиатуры, к которой мы сейчас и перейдем.

4.4. ВВОД С КЛАВИАТУРЫ, ИЛИ ЧЕГО ЖДЕТ ОТ НАС ПРОГРАММА

Самая простая команда ввода с клавиатуры выглядит так:
получить __ символ (имя)

где в скобках стоит имя переменной символьного типа. При выполнении этой команды программа приостанавливается и ждет, пока с клавиатуры не введут какой-нибудь символ. После этого выполнение программы продолжается, а значением соответствующей переменной становится введенный символ. Вот пример программы, которая вводит с клавиатуры символ, а затем печатает его на экране дважды:

ПРОГРАММА 3

а : симв

получить __ символ (а)

напечатать (а, а)

КОНЕЦ ПРОГРАММЫ

Команда «получить __ символ» — неременная часть всех диалоговых пакетов. Убедиться в этом можно экспериментально. Запустите какую-нибудь диалоговую программу. Что произойдет? В течение некоторого времени на экране будет рисоваться та или иная картинка, после чего всякая внешняя «жизнь» ПК прекратится. Что сейчас делает находящаяся в ПК

прикладная программа? Она приостановилась, выполняя команду «получить __ символ», потому что мы пока не прикасались к клавиатуре. Если нажать одну из клавиш, которые предлагает нажать появившееся на экране меню, программа выполнит соответствующее действие, после чего опять застынет на команде «получить __ символ» и т. д.

Хотелось бы, однако, вводить значения переменных и иных типов, кроме символьного. Проще всего это сделать при помощи команды, которую мы запишем так:

вести (имя)

где в скобках указывается имя переменной, которая может быть любого типа. По этой команде программа останавливается и ждет, пока пользователь не введет с клавиатуры соответствующее значение (например, целое число, если переменная целая). В процессе ввода каждый вводимый символ немедленно отображается на экране (чего не происходит при работе команды «получить символ»), т. е. команда «вести» управляет и клавиатурой и экраном. Заканчивается ввод нажатием клавиши «Возврат каретки».

Обратите внимание на то, что команды ввода, в отличие от команд вывода на экран, *изменяют* значения переменных — тех, чьи имена записаны в скобках после названия команды.

Введенных команд уже достаточно, чтобы писать несложные программы. Напишем, например, программу, которая вводит с клавиатуры число и выводит на экран его квадрат:

ПРОГРАММА 4

X : цел

перевести __ строку

напечатать («Введите целое число»),

вести (X);

перевести __ строку

напечатать («Квадрат числа», X, « равен », $X * X$),

КОНЕЦ ПРОГРАММЫ

Все же программы, которые мы конструировали, чересчур «прямолинейны»: они требуют последовательно выполнить записанные между словами ПРОГРАММА и КОНЕЦ ПРОГРАММЫ команды — лишь те, что записаны, и ровно столько, сколько записано, независимо ни от чего. Наблюдение за работой уже существующих прикладных программ на ПК подсказывает нам, что те устроены более гибко. Во-первых, действия, выполняемые ими, существенно зависят от наших действий во время диалога или от чего-то еще (от того, например, найден ли на диске файл с заданным именем, подключен ли принтер, на котором приказано что-то распечатать, нашлись ли у решаемого уравнения действительные корни и т. д.). Во-вторых, при выполнении некоторых наших просьб компьютер надолго задумывается — а значит успевает выполнить за это время десятки тысяч команд. Неужели все они записаны в программе последовательно?

Оказывается, последовательная запись команд — это не единственный способ сконструировать из них программу. Пришло время рассказать и о других способах.

4.5. КАК МЫ КОНСТРУИРУЕМ ПРОГРАММЫ

Надеемся, что уже из приведенных выше примеров понятно, что является обычной задачей программиста. В общем виде она звучит как-то так: требуется объяснить кому-то (человеку, устройству и т. п.), как достичь той или иной цели. При этом известно, что инструктируемый умеет выполнять лишь несколько простых действий. Вот несколько типично программистских задач, которые каждый из нас решает ежедневно.

1. Вы сидите в автомобиле. Перед Вами ключ зажигания, рычаг коробки передач, педали сцепления и газа. Как надо действовать, чтобы стронуть машину с места?

2. Объясните, как добраться до Вашего дома от центра города.

3. Имеется пустая кастрюля, вода, крупа, соль и газовая плита. Как приготовить кашу?

В задаче 1 элементарными действиями (или элементарными командами), видимо, являются нажатия на педали и повороты рычагов и ручек. В задаче 2 фразы вроде «дойди до конца улицы», «поверни направо», «сядь в автобус номер сто тридцать», тоже скорее всего не потребуют дополнительного пояснения. Но как из них составляется полный текст объяснения? Приведем три примера:

а) «Дойди до конца улицы, затем поверни налево».

б) «Если на остановке стоит автобус, садись в него, иначе иди пешком».

в) «Пропускай все перекрестки до тех пор, пока не встретишь перекрестка со светофором».

Эти три примера иллюстрируют три общепринятых способа конструирования программ из команд. Способом а) мы уже воспользовались при написании программ, когда требовали последовательно выполнить несколько элементарных команд. Предложение б) собрано из двух команд («поезжай на автобусе» и «иди пешком») и одной проверки истинности условия («Есть ли автобус на остановке?») при помощи так называемой *конструкции выбора*. Мы будем записывать эту конструкцию так:

ЕСЛИ есть автобус ТО
поехать на автобусе

ИНАЧЕ

идти пешком
КОНЕЦ ЕСЛИ

Такая конструкция требует от того, кто исполняет программу, сначала выполнить записанную между ЕСЛИ и ТО проверку и, в зависимости от ее результата, выполнить либо одно, либо другое действие. Слова ЕСЛИ, ТО, ИНАЧЕ, КОНЕЦ ЕСЛИ не являются командами, а лишь отделяют одни части конструкции выбора от других, т. е. по сути дела играют роль знаков препинания в предложении. Такие или похожие слова-знаки есть в любом языке программирования — их называют *ключевыми словами* или (реже) *иероглифами*.

Рассмотрим третий пример. Мы будем формально записывать соответствующую ему конструкцию так:

ПОКА на перекрестке нет светофора ПОВТОРЯТЬ
 дойти до следующего перекрестка
КОНЕЦ ПОКА

Эта конструкция, называемая *конструкцией цикла*, требует выполнять записанное между ключевыми словами ПОВТОРЯТЬ и КОНЕЦ ПОКА действие (это действие называется *телом цикла*), пока истинно записанное между ПОКА и ПОВТОРЯТЬ условие. Несколько отличительных особенностей этой конструкции.

Нельзя заранее сказать, сколько раз будет выполнена команда «дойти до следующего перекрестка» — это зависит от внешних условий (в данном случае, от того, встретится ли на дороге светофор). Например, если исполнитель этой программы уже стоит на перекрестке со светофором, т. е. условие выполнено с самого начала, то тело цикла не выполнится ни разу.

Можно построить конструкцию цикла, которая будет выполняться бесконечно долго. Например, выполнение программы

ПОКА на перекрестке нет светофора ПОВТОРЯТЬ
 повернуть налево
 дойти до следующего перекрестка
КОНЕЦ ПОКА

не прекратится никогда, если в соответствующем квартале города нет светофоров. Программисты говорят: программа заикливается. Конечно, правильно составленная конструкция не должна заикливаться.

Условие, записанное после ПОКА, проверяется только перед очередным выполнением тела цикла. Если условие станет истинным уже где-то посередине, тело цикла все равно выполнится до конца. Например, если программа

ПОКА кастрюля не полна ПОВТОРЯТЬ
 долить стакан воды
 долить стакан воды

КОНЕЦ ПОКА

применяется к пустой кастрюле, вмещающей ровно 5 стаканов воды, то один стакан прольется на пол.

Именно благодаря конструкции цикла можно с помощью маленькой программы заставить ее исполнителя выполнить большое число действий.

ДЛЯ переменная := начальное __ значение ДО конечное __
значение ШАГ такой-то ПОВТОРЯТЬ

КОНЕЦ ДЛЯ

КОНЕЦ ДЛЯ

Пример 2. Напишем программу, которая вводит несколько чисел и печатает максимальное. Как это сделать? По-прежнему заведем переменную *тах* и будем вводить с клавиатуры числа по одному, всякий раз устанавливая *тах* равным максимуму из уже введенных чисел. Будем действовать так, пока не введем все числа. Программа должна выглядеть как-то так:

```
установить тах равным
максимальному из введенных чисел
ПОКА есть числа ПОВТОРЯТЬ
    ввести число
    установить тах равным
    максимальному из введенных чисел
КОНЕЦ ПОКА
```

Каким надо сделать *тах* вначале, ведь еще не введено ни одного числа? Другими словами, чему равен максимум из пустого множества чисел? Попытаемся обойти эту трудность, договорившись, что на вход программе подается хотя бы одно число. Тогда можно сначала ввести одно число, а затем сделать *тах* равным ему:

```
ввести ( x )
тах := x
```

Теперь *тах* стал равным максимуму из введенных чисел (введено-то было только одно число). Будем теперь вводить числа, пока они не кончатся, меняя переменную *тах* так, чтобы она оставалась равной максимуму из введенных чисел. Пусть введено еще одно число. На рис. 4.2 показано, как изменится ситуация в этом случае. Будет ли *тах* по-прежнему максимальным из введенных чисел? Да, будет, если введенное только что число не превышает *тах*. Если оно больше текущего значения *тах*, надо сделать *тах* равным ему. Таким образом, тело цикла можно построить так:

```
ввести ( x )
тах := x
ЕСЛИ тах < x ТО
    тах := x
КОНЕЦ ЕСЛИ
```

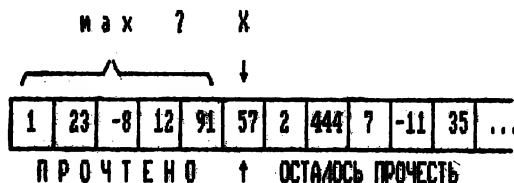


Рис. 4.2. К программе вычисления максимума

Как видим, конструкция ЕСЛИ получилась усеченной — отсутствует вариант ИНАЧЕ, поскольку если x превышает max , то делать ничего не нужно.

Осталось научиться проверять, что число не последнее. Однако без дополнительной информации сделать этого нельзя — пользователь должен как-то сообщить программе, что числа кончились. Ясно, что это можно сделать разными способами. Например, можно договориться, что пользователь вначале сообщает, сколько чисел будет введено. Другой способ — после ввода каждого из чисел сообщать программе, будут ли еще. Мы так и поступим: пусть после того, как пользователь ввел очередное число, программа спрашивает его «Это все?» и в зависимости от ответа либо продолжает либо прекращает работу. Окончательная программа тогда станет такой:

ПРОГРАММА 6

x, max : цел

ответ : симв

перевести __ строку

напечатать («Введите число:»)

вести (x)

перевести __ строку

напечатать («Это все? (д/н)»)

вести (ответ)

$max := x$

ПОКА ответ = 'н' ПОВТОРЯТЬ

перевести __ строку

напечатать («Введите число:»)

вести (x)

перевести __ строку

напечатать («Это все? (д/н)»)

вести (ответ)

ЕСЛИ $max < x$ ТО

$max := x$

КОНЕЦ ЕСЛИ

КОНЕЦ ПОКА

перевести __ строку

напечатать («Максимальное из введенных чисел — », max)

КОНЕЦ ПРОГРАММЫ

Диалог с ПК, работающим по этой программе, будет происходить так: программа напечатает «введите число» и будет ждать, пока на клавиатуре не наберут какое-нибудь число. Когда это будет сделано и пользователь нажмет клавишу «Возврат каретки», программа спросит его «Это все?». Если пользователь наберет «н» и затем даст возврат каретки, программа попросит его ввести еще одно число. Если же будет набран какой-нибудь другой символ, программа посчитает ввод чисел законченным и выведет на экран максимальное из введен-

ных пользователем чисел. Типичный диалог при работе с этой программой может быть, например, таким (мы обозначаем приглашение операционной системы знаком «С : > »):

```
С : > запустить программу
Введите число : 12
Это все? (д/н) : н
Введите число : —7
Это все? (д/н) : н
Введите число : 5
Это все? (д/н) : д
Максимальное из введенных чисел : 12
С : >
```

З а м е ч а н и е. Мы не расшифровываем пока, ни что означает запись «запустить программу» в первой строке этого диалога, ни того, как получить эту выполняемую программу из приведенного текста. Заметим только, что помимо необходимости перевода на компьютерный язык программирования, прогон на конкретной машине может потребовать изменить и форму диалога. Это связано с тем, что не все машины способны печатать на экране и вводить с клавиатуры русские буквы.

Упражнение. Сколько раз выполнится тело цикла в последнем примере, если на первый же вопрос «Это все?» пользователь ответит утвердительно (т. е. «д»)? Получится ли правильный результат?

Упражнение. Примем другой стандарт на диалог. Пусть программа сначала спрашивает, сколько будет чисел. Тогда необязательно выяснять про каждое из вводимых чисел, не последнее ли оно. Измените программу так, чтобы она поддерживала такой диалог.

Решение. Заведем целую переменную с именем «осталось» и будем хранить в ней количество чисел, которое осталось ввести. Тогда программа станет такой:

ПРОГРАММА

```
х, тах : цел
осталось : цел
перевести __ строку
напечатать («Сколько будет чисел? :»).
вести (осталось)
перевести __ строку
напечатать («Введите число :»)
вести ( х ); осталось := осталось — 1
тах := х
```

ПОКА осталось $\neq$ 0 ПОВТОРЯТЬ

```
    перевести __ строку
    напечатать («Введите число :»)
    ввести ( х ); осталось := осталось — 1
ЕСЛИ тах < х ТО
    тах := х
```

КОНЕЦ ЕСЛИ КОНЕЦ ПОКА

перевести — строку
напечатать («Максимальное из введенных чисел — »,
max)

КОНЕЦ ПРОГРАММЫ

Упражнение. Предыдущая программа предполагает, что в ответ на вопрос «Сколько будет чисел?» пользователь введет натуральное число (т. е. единицу или больше). Что произойдет, если пользователь по рассеянности или из вредности введет ноль или минус один?

Как видим, программы получаются уже довольно сложными — они едва умещаются на странице и разом понять, что делает та или иная программа, уже не так просто. Как же программисты-профессионалы ухитряются писать правильно работающие программы длиной в десятки тысяч строк? Вопрос интересный и непростой — слишком непростой, чтобы достаточно полно ответить на него в этой книге. Мы лишь перечислим несколько приемов программирования, доступных при работе на любом компьютерном языке и делающих программу более читаемой.

Полезным приложением к любой из Ваших программ будет инструкция по ее использованию. Например, такая: «Данная программа определяет максимальное из введенных пользователем с клавиатуры целых чисел. После запуска программы пользователь должен ввести необходимое количество чисел (не меньше одного), всякий раз отвечая на вопрос «Это все?» клавишей «н» с последующим возвратом каретки, если предполагается вводить еще числа. Как только он ответит «д» на этот вопрос, программа выведет на экран максимальное из введенных чисел и закончит работу. Программа хранится в файле с именем...».

Теперь Вашей программой сможет воспользоваться кто-то еще. Полезно записать такую инструкцию в файл на диске в тот же каталог, в котором находится программа, либо собрать все такие описания в специально предназначенный для этого каталог. Стоит, однако, позаботиться и о человеке, которому, быть может, придется читать или менять исходный текст программы. Скорее всего, этим читателем окажетесь через неделю-другую Вы сами, а уж о себе-то тем более следует позаботиться. Удобней всего сделать необходимые пояснения прямо в тексте программы, например, так:

ПРОГРАММА 6

х, max : цел
{последнее и максимальное из введенных чисел, соответственно}

ответ : симв
{принимает значение «д», если ввод чисел не закончен}

```

перевести __ строку
напечатать («Введите число : »)
вести ( x )
перевести __ строку
напечатать («Это все? (д/н)»)
вести (ответ)
max := x
{max = максимальное из всех введенных чисел}
{ответ = «н», если есть еще числа}
ПОКА ответ = ' н ' ПОВТОРЯТЬ
    {max = максимальное из всех введенных чисел}
    {есть еще числа}
    перевести __ строку
    напечатать («Введите число : »)
    вести ( x )
    перевести __ строку
    напечатать («Это все? (д/н)»)
    вести (ответ)
    {max = максимальное из всех введенных чисел,
                                     не считая последнего}
    {x = последнее введенное число}
    ЕСЛИ max < x ТО
        max := x
    КОНЕЦ ЕСЛИ
    {max = максимальное из всех введенных чисел}
    {ответ = «н», если есть еще числа}
КОНЕЦ ПОКА
{max = максимальное из всех введенных чисел}
{чисел больше нет, т. е. max = максим. из всех}
перевести __ строку
напечатать («Максимальное из введенных чисел — »,
                                     max)

```

КОНЕЦ ПРОГРАММЫ

Мы используем фигурные скобки для выделения комментария из прочего текста программы. Любой язык программирования дает возможность записывать комментарии прямо в тексте программы. Приходится только отделять эти комментарии от самой программы специальными символами, чтобы компьютер понял, что это информация не для него, и пропустил ее. Что должно быть написано в комментариях? Вот несколько советов.

1. Комментируйте описания переменных, поясняя, *что* хранит эта переменная во время работы программы. Если вам заранее известно, какие именно значения будет принимать переменная, то перечислите их. Например, может оказаться, что какая-то целая переменная принимает лишь значения ноль и единица, а другая целая переменная не может быть отрицательной и т. п. — обязательно отмечайте это в комментариях!

2. Старайтесь комментировать *не действия*, которые будет выполнять компьютер, а *ситуации*, в которых он (по Вашему мнению) должен в соответствующий момент оказаться. Действительно, вряд ли стоит команду «перевести строку» снабжать комментарием «здесь происходит переход на следующую строку». В то же время, эволюция смысла переменной тах в последней программе не так очевидна (например, ее значение не в любой момент есть максимум из всех введенных чисел). Поэтому крайне важно проследить за ее изменениями до самого последнего момента — ведь в конце концов в ней оказывается результат работы программы. Кстати, из последнего комментария легко понять, что же вычисляет эта программа.

3. Будьте кратки — ведь помимо своей прогрессивной роли комментарии удлиняют текст программы, так что чрезмерная словоохотливость комментатора может, наоборот, затруднить понимание. Относитесь к написанию грамотного комментария не менее серьезно, чем к написанию правильной программы.

На читаемость программы влияет и геометрия текста. Читая книгу, мы не воспринимаем красные строки как нарушение связности текста — конечно, если текст удачно разбит на абзацы. Более того, попробуйте прочесть страницу текста, целиком заполненную символами — безо всяких красных строк и перебивок. Наверняка Вы почувствуете усталость, а может быть, и упустите что-нибудь существенное. Не менее важна и геометрия текста программы при ее чтении человеком. Для компьютера она несущественна, так что большинство языков программирования допускают произвольное расположение текста по строкам. Поэтому в наших силах записывать программу так, чтобы геометрия текста как-то облегчала понимание. Единого рецепта тут нет — мы, например, пользуемся отступами, помещая окончание любой конструкции в точности под ее началом, а внутренний текст сдвигая на несколько позиций вправо. На наш взгляд, это помогает понять структуру программы. Например, ответ на вопрос «Где кончается тело этого цикла?» не потребует изучения всего этого тела — достаточно лишь скользнуть взглядом вниз по странице и найти слово КОНЕЦ ПОКА, расположенное в точности под соответствующим ПОКА.

Не исключено, что Вы придумаете более удачное, на Ваш взгляд, расположение. Самое главное — не экономьте строки, избегайте писать весь текст подряд, если Вашей целью не является зашифровка программы в доступную лишь компьютерному пониманию форму. Вот, например, что можно сделать с нашей программой, не нарушая придуманных нами правил записи: ПРОГРАММА 6; x, тах : цел; ответ : симв; перевести — строку; напечатать («Введите число:»); ввести (x); перевести — строку; напечатать («Это все? (д/н)»); ввести (ответ); тах := x; ПОКА ответ = 'н' ПОВТОРЯТЬ; перевести —

строку; напечатать («Введите число:»); ввести (х); перевести __ строку; напечатать («Это все? (д/н)»); ввести (ответ); ЕСЛИ тах<х ТО тах = х КОНЕЦ ЕСЛИ КОНЕЦ ПОКА; перевести __ строку; напечатать («Максимальное из введенных чисел — », тах); КОНЕЦ ПРОГРАММЫ

Но стоит ли так поступать?

4.6. ПРОЦЕДУРЫ, ИЛИ СПОСОБ ОПРЕДЕЛЕНИЯ НОВЫХ КОМАНД

Ценность приведенных выше назиданий растет с ростом размера программы, которую Вы пишете. Вряд ли правильное расположение текста так уж важно в программе вычисления максимума, если всю ее можно без труда выучить наизусть. Когда длина программы растет, понимать ее (а тем более менять), становится все труднее. Конечно, удачное размещение текста (отступы и т. п.) упрощает понимание и отодвигает тот драматический момент, когда программа начнет буквально разваливаться на части в голове своего создателя, а что она сделает, попав в компьютер, уже не сможет предсказать никто. Но когда-то этот момент наступает все равно. Связано это просто с размерами текста — попробуй-ка, пойми, что делает тело цикла, если оно начинается на этой странице, а кончается через одну — тут уж никакие отступы не спасут. Конечно, выручает продуманный комментарий (не пренебрегайте им в ваших программах!), но в большой программе и комментариев много, так что выудить нужный не всегда просто. Можно спорить о том, когда именно начинаются такие беды. Авторы этих строк, например, не способны разобраться в программах (даже в собственных), если они не умещаются целиком на экране дисплея (или на одном листочке бумаги). Как же быть, если программа не получается такой короткой?

А почему программа получается длинной? Видимо, потому, что стандартные команды, которые мы используем, слишком слабы. Представим себе, что в нашей программе вычисления максимума мы могли бы использовать такую команду:

ввести __ число (имя1, имя2)

которая выводит на экран просьбу ввести число, затем вводит число и ответ на вопрос, есть ли еще числа, помещая число и ответ в переменные, чьи имена указаны в скобках после команды. Тогда программа станет куда короче:

ПРОГРАММА

х, тах : цел

ответ : симв

ввести __ число (х, ответ)

тах := х

ПОКА ответ = 'н' ПОВТОРЯТЬ

ввести __ число (х, ответ)

```

ЕСЛИ max<x ТО
    max := x
КОНЕЦ ЕСЛИ
КОНЕЦ ПОКА
перевести — строку
напечатать («Максимальное из введенных чисел — »,
max)
КОНЕЦ ПРОГРАММЫ

```

Большинство языков программирования дают возможность придумать новую команду и использовать ее в своих программах наряду со стандартными. Нужно только объяснить компьютеру, как выражается эта новая команда через уже существующие. Мы будем записывать это объяснение так:

```

ПРОЦЕДУРА ввести — число (число: вых цел, буква: вых
симв)

```

```

    перевести — строку
    напечатать («Введите число:»)
    ввести (число)
    перевести — строку
    напечатать («Это все? (д/н)»)
    ввести (буква)

```

```

КОНЕЦ ПРОЦЕДУРЫ

```

Поясним, что здесь написано. Верхнюю строку, называемую *заголовком процедуры*, следует понимать так: сейчас будет расшифрована употребленная в программе команда «ввести число». При записи этой команды в программе следует всякий раз указывать в скобках после команды имена двух переменных. Первая из них должна быть целой, вторая — символьной. Обе эти переменные являются *выходными*, т. е. они получают новые значения в результате выполнения команды «ввести число», а их значения до выполнения этой команды ни на что не повлияют. Именно это мы имели в виду, написав «вых» после каждой переменной в заголовке. Дальнейший текст имеет такой смысл: всякий раз, когда программе требуется выполнить команду «ввести число» с какими-либо двумя переменными, скажем, «х» и «ответ», надо сделать с этими переменными все то, что написано здесь (т. е. после заголовка процедуры) про переменные «число» и «буква».

Текст между ключевыми словами ПРОЦЕДУРА и КОНЕЦ ПРОЦЕДУРЫ называют *описанием процедуры* (в данном случае процедуры «ввести число»). Каждое использование команды «ввести число» в программе называют обращением к процедуре или *вызовом процедуры*. Придуманы специальные названия и для того, что пишется в скобках после имени процедуры. Переменные «число» и «буква» называют *формальными параметрами* процедуры, а те переменные, которые пишутся в обращении к процедуре — *фактическими параметрами* данного обращения. Программа может обращаться к процедуре

несколько раз, при этом фактические параметры могут быть разными — лишь бы их типы были те же, что и у соответствующих формальных параметров (в данном случае целый и символьный). Например, предпримем очевидную оптимизацию в программе вычисления максимума (странно, что мы сразу о ней не подумали):

ПРОГРАММА

х, тах : цел

ответ : симв

вести __ число (тах, ответ)

ПОКА ответ = 'н' ПОВТОРЯТЬ

вести __ число (х, ответ)

дальше все по-старому...

КОНЕЦ ПРОГРАММЫ

При первом обращении к процедуре «вести число» все то, что написано про формальный параметр «число» в описании процедуры, в действительности произойдет с переменной тах. При втором обращении (это то, которое находится внутри цикла) то же самое произойдет с переменной «х».

А вот переменные «буква» и «число» в программе употреблять нельзя — они имеют смысл лишь внутри описания процедуры. Да это, строго говоря, и не переменные вовсе, а скорее «тени» переменных — ведь конкретные действия все равно выполняются над переменными программы.

С чем-то похожим на параметры мы встречаемся, читая тексты вроде «Пассажир имеет право...» или «Уважаемые читатели нашей библиотеки, просим вас...». Во всяком правиле подобного рода присутствует некий собирательный образ (параметр): пассажир, читатель и т. п. Кто это конкретно — выясняется лишь при применении правила. Продолжая аналогию, заметим, что всегда — явно или неявно — оговаривается тип параметра, т. е. множество объектов, к которым данное правило применимо: скажем, только читатель, причем только данной библиотеки.

Наряду с выходными параметрами, процедуры допускают и параметры *входные*. Рассмотрим, например, такое описание:

ПРОЦЕДУРА вычислить __ гипотенузу (к1, к2 : вх вещ, гип: вх вещ)

гип : = корень (к1 * к1 + к2 * к2)

КОНЕЦ ПРОЦЕДУРЫ

Мы считаем, что функция «корень» является стандартной, т. е. ее можно употреблять в арифметическом выражении. Первые два параметра не изменяются в результате обращения к процедуре «вычислить гипотенузу». Напротив, их значения используются при этом вычислении, т. е. в основной программе допустимо, например, такое обращение к этой процедуре:

вычислить — гипотенузу (3.0, 4.0, гипотенуза)

в результате которого переменная «гипотенуза» станет с точ-

ностью несколько знаков после запятой равной пяти. (Приблизительность результата связана с возможными ошибками из-за неточности представления вещественных чисел в машине.)

Есть еще один вариант описания процедуры, если она вычисляет лишь одно значение. Скажем, процедуру «вычислить гипотенузу» можно реализовать и так:

ФУНКЦИЯ гипотенуза (k_1, k_2 : вещ) : вещ

ОТВЕТ (корень ($k_1 * k_1 + k_2 * k_2$))

КОНЕЦ ФУНКЦИИ

В программе, к которой добавлено такое описание, становятся допустимыми команды вроде:

результат := гипотенуза ($a + 2.5, 12$) * 4

т. е. мы получили право употреблять функцию «гипотенуза» в арифметических выражениях, подобно тому, как мы можем употреблять функцию «синус» или какую-либо еще. Перечислим отличия описаний функций от описаний процедур.

Функция возвращает лишь один результат. Тип этого результата должен быть указан в заголовке описания функции после списка формальных параметров. Функция не имеет права менять никакие переменные программы, т. е. все параметры функции — входные и слово «вх» поэтому опускается.

После заголовка функции могут быть написаны любые команды по обычным правилам. Однако в конце текста описания функции должна быть специальная команда «ответ», в скобках после которой записывается выражение, значение которого функция должна возвратить в качестве результата. Тип этого выражения должен соответствовать указанному в заголовке функции.

Помимо экономии бумаги при записи полного текста программы (и памяти при ее выполнении) процедуры хороши тем, что позволяют явно разделить задачу на более простые подзадачи. Кроме того, есть и некоторые организационные преимущества. Например, программу может писать один человек, а описание использованной там процедуры — другой, нужно лишь, чтобы они договорились, что получает эта процедура (входные параметры) и что она должна вычислить (выходные параметры). Более того, ПК позволяет писать программу на одном языке программирования, а вызываемую ей процедуру — на другом! Все языки программирования имеют так называемый *набор стандартных процедур* — это означает, что их описания (чаще всего написанные в кодах машины) поставляются вместе с компилятором с соответствующего языка, и их можно вызывать из любых программ, написанных на этом языке. Такова, к примеру, функция «корень», которую мы только что использовали. Не исключено, что некоторые из тех команд, о которых мы говорили («получить символ» и т. п.), являются в каком-нибудь языке вызовами процедур. Описания всех стандартных для данного языка процедур обычно собраны

в специальных файлах, называемых *библиотеками*. Пользователь имеет возможность создавать свои библиотеки из описаний разработанных им процедур, расширяя таким образом набор допустимых команд языка (если считать вызовы разных процедур разными командами).

Программы, рассмотренные в этой главе, могут быть в принципе выполнены на любом, а стало быть, и на персональном компьютере — осталось только перевести их на компьютерный язык и ввести в машину. В гл. 6 мы рассмотрим перевод всех шести построенных нами программ на различные компьютерные языки. Программирование ПК, однако, имеет ряд специфических особенностей, рассказу о которых мы посвящаем следующую главу.

Таким образом, если Вы испытываете желание немедленно выполнить на ПК хоть какую-нибудь программу, можно пропустить следующую главу, взявшись сразу за гл. 6. А к гл. 5 стоит вернуться тогда, когда описанный выше минимум программных средств покажется Вам недостаточным.

Глава 5. ПРОГРАММИРОВАНИЕ ДЛЯ ПЕРСОНАЛЬНОЙ ЭВМ

В этой главе мы не станем более распространяться об общих приемах программирования, а расскажем об особенностях программирования для ПК. Что же нового внес ПК в программирование? С принципиальной точки зрения, почти ничего. Персональный компьютер «понимает» те же языки программирования, что и большие ЭВМ, выполнение написанной программы требует примерно тех же операций, что и раньше (ввод текста, компиляция, редактирование связей). Изменения же сводятся вот к чему:

работавший на ПК — монополюный владелец всей ЭВМ. У ПК нет защищенных операций, как это бывает в больших ЭВМ. В результате вы можете заставить ПК сделать все, на что он в принципе способен. В частности, пользовательская программа может сделать все то, что умеет делать операционная система, и мы обязательно расскажем, как этого добиться;

в комплект поставки любой ЭВМ входит набор внешних устройств (дисплей, печатающее устройство и т. д.). Для ПК также есть стандартный (или, во всяком случае, типичный) набор внешних устройств, управление которыми осуществляется несколько иначе, чем на большой ЭВМ. О том, что это за устройства, и как программа может ими управлять, мы также расскажем.

Итак, в этой главе мы попытаемся описать возможно более полный набор процедур для управления персональным компьютером, ориентируясь в основном на компьютеры типа IBM PC.

Предупреждаем — данная глава, в отличие от предыдущих, рассчитана на читателей, имеющих опыт работы на какой-либо ЭВМ (или приобретших его в процессе работы на ПК). В частности, мы будем использовать ряд терминов — байт, массив, адрес и т. п. — не объясняя или почти не объясняя, что это такое.

Описания всех вводимых ниже команд будут приводиться в таком же виде, в каком мы писали заголовки процедур в предыдущей главе. После имени команды в скобках указывается список параметров в форме «имя: вх/вых тип». Наряду с типами, описанными в гл. 4, могут использоваться и другие. Например, встречается тип «байт», которому соответствуют восьмиразрядные двоичные числа (т. е. числа от 0 до 255 в десятичной записи). Заметим, что символы в машине фактически закодированы такими же числами, так что типы «байт» и «симв» эквивалентны. Однако мы используем и слово «симв», если передаваемый параметр по смыслу является символом.

5.1. УПРАВЛЕНИЕ ОПЕРАЦИОННОЙ СИСТЕМОЙ

Основное назначение ОС — поддержание работы с деревом файлов. Как Вы, видимо, поняли из предыдущих глав, большинство команд ОС — это именно команды работы с файлами. Заметим, однако, что с точки зрения ОС, файлы являются неделимыми объектами, которые можно создавать, копировать, уничтожать, переименовывать и т. п., но нельзя изменять их содержимое — ОС как бы вообще не знает, из чего состоит отдельный файл. Напротив, программу пользователя чаще всего интересует хранящаяся в файле информация, либо она наполняет файл новой информацией. Посмотрим, как организуется такая работа с файлом в программе. Ясно, что для этого нужны команды, позволяющие записывать информацию из программы в файл, и команды, читающие информацию из заданного файла в какие-либо переменные программы. То есть команда вида

записать (куда: имяфайла, что: информация)
кажется приемлемым вариантом. На самом деле такая команда работала бы слишком медленно. Действительно, представим себе, что программа вырабатывает интересующую нас информацию небольшими порциями (как правило, так и есть) и всякий раз по выработке очередной порции записывает ее в файл. Тогда при каждом выполнении команды записи приходится осуществлять, по крайней мере, следующие операции:

1) получив имя файла, найти сам файл, т. е. определить, на каком носителе информации должен располагаться этот файл, опросить этот носитель на предмет того, есть ли там файл с таким именем, понять, в каком именно месте на носителе этот файл расположен;

2) подготовиться к записи информации: быть может, включить мотор, подвести записывающее устройство к нужному месту на носителе и т. д.;

3) записать информацию.

Все эти действия отнимают довольно много времени. При этом ясно, что некоторые из них можно выполнить один раз — перед началом работы с файлом. С этой целью предусматривается специальная команда начала работы с файлом, которую мы запишем так:

открыть __ файл (какой: вх строка, зачем: вх, кодфайла: вых файл, удалось: вых лог)

Первый параметр этой команды — строка, содержащая путь к соответствующему файлу, записанная по тем же правилам, что и при диалоге с ОС (нельзя только использовать звездочки и вопросительные знаки). По этой команде система выполняет ряд подготовительных операций по работе с файлом, несколько различающихся в зависимости от параметра «зачем». Тип этого параметра различен в разных языках, а смысл такой — он задает способ последующего использования файла. Вот чаще всего применяемые значения этого параметра и соответствующие им действия:

ДЛЯ ЧТЕНИЯ — ищется файл с заданным именем. В дальнейшем этот файл будет только читаться, причем последующие команды чтения будут брать информацию, начиная с самого начала файла;

ДЛЯ ЗАПИСИ — файл с заданным именем делается пустым, если он существует, и создается, если такого файла нет. Последующие команды записи в файл добавляют в него информацию начиная с начала. Таким образом, все содержимое файла будет создано заново;

ДЛЯ ДОБАВЛЕНИЯ — то же, что и для записи, только старое содержимое файла не стирается и новая информация записывается в конец.

Если описанные операции выполнены успешно, в переменную, написанную на месте третьего параметра в команде «открыть», помещается так называемый код открытого файла. Именно этот код, а не имя файла, нужно будет указывать при последующей работе с файлом командами записи и чтения. В некоторых языках для хранения кода файла специально предусмотрены переменные типа «файл». Может, однако, случиться так, что открыть файл указанным способом не удалось — например, он уже был открыт, или открываемый для чтения файл не существует, или открытый для записи файл защищен от изменения и т. п. Тогда код файла не вырабатывается, а логическая переменная, записанная на месте последнего параметра, получает значение ЛОЖЬ. При успешном открытии она получит значение ИСТИНА. Подобные неудачи возможны при выполнении практически любых команд работы с файловой

системой, так что последним параметром во многих командах мы будем писать слово «удалось», уже никак его не комментируя.

Команды работы с открытым файлом выглядят так:

прочитать __ файл (откуда: вх файл, сколько: вх цел, куда: вых массив, удалось)

записать __ в __ файл (куда: вх файл, сколько: вх цел, откуда: вх массив, удалось)

Первым параметром обеих команд должен быть код открытого нужным образом файла. Вторым параметр задает число байтов, которое следует прочитать или записать. На месте третьего параметра должно быть имя массива, содержащего по крайней мере заданное предыдущим параметром количество байтов. В случае команды записи это может быть просто строка соответствующей длины, рассматриваемая как массив символов.

Если файл уже открыт, например для чтения, открыть его для записи нельзя. Если же мы закончили чтение и хотим что-либо записать в этот файл, его необходимо закрыть при помощи команды:

закрыть __ файл (код: вх файл)

Теперь файл с заданным кодом перестал быть открытым, т. е. ему больше не соответствует никакой код, так что ни чтение из него, ни запись в него невозможны. При необходимости, однако, его можно снова открыть в любом режиме. Заметьте, что при повторном открытии ему вовсе не обязательно будет соответствовать тот же код. Файл рекомендуется закрывать, если Вы не собираетесь больше с ним работать. Впрочем, по окончании работы программы, ОС сама закроет все Ваши незакрытые файлы. Приведем фрагмент программы, который записывает в файл TEXT.TXT, находящийся в каталоге MYDIR на диске C, слово «новость»:

текст : файл

удалось : лог

открыть («C:\MYDIR\TEXT.TXT», ДЛЯ ЗАПИСИ, текст, удалось)

ЕСЛИ удалось ТО

записать __ в __ файл (текст, 7, «новость», удалось)

закрыть __ файл (текст)

КОНЕЦ ЕСЛИ

ЕСЛИ не удалось ТО

напечатать («Ошибка при записи в файл!»)

КОНЕЦ ЕСЛИ

Описанный способ обмена информацией с файлом принято называть бесформатным, так как файлы и записываемые в них данные рассматриваются при этом как последовательность байтов. Есть иная возможность: записывать информацию в файл при помощи команды «напечатать», и считывать ее оттуда командой «ввести». В ряде языков для этого достаточно добавить код открытого файла в список параметров этих команд.

Пример:

код: файл
удалось: лог
х: вещ

открыть — файл («RESULT.DAT», ДЛЯ ЗАПИСИ, код,
удалось)
напечатать (код, «Вычисленный результат равен», х)

Записанный командами «напечатать» файл можно затем просмотреть командой TYPE ОС, числа в нем будут представлены в привычной десятичной записи. Правда, при этом они займут в файле больше места, чем занимали в памяти ПК. Команды бесформатного обмена предпочтительнее при записи больших числовых массивов, если предполагается, что сформированный файл будет читаться лишь при помощи программ.

Перейдем теперь к командам ОС, позволяющим опрашивать или изменять дерево файлов либо параметры отдельных файлов. Сначала поговорим об отдельном файле. На каждом носителе информации, скажем на любой дискете или на жестком диске, часть места отведена для справочной информации: там описано, как устроено дерево файлов, есть специальное «досье» и на каждый файл. Так, про каждый имеющийся на носителе файл хранятся данные о размере, дате и времени его последнего изменения, а также набор флагов типа «да»/«нет», отвечающих на такие вопросы о файле (в скобках указана мнемоника, которую мы будем использовать при обращении):

- а) Это каталог? (КАТАЛОГ),
- б) Файл защищен от изменения? (ЗАЩИЩЕН),
- в) Это скрытый файл? (СКРЫТЫЙ),
- г) Это архивный файл? (АРХИВНЫЙ),
- д) Это системный файл? (СИСТЕМНЫЙ),
- е) Это метка тома? (МЕТКА).

Свойства г) — е) мы не будем обсуждать. Заметим лишь, что обычно файл является архивным, но не является ни системным, ни меткой тома. Параметры б) — е) имеют смысл, лишь если файл не является каталогом. О защите от изменения мы говорили в гл. 3, так что осталось лишь сказать несколько слов о скрытых файлах. Они отличаются от обычных лишь тем, что «не видны» операционной системе при некоторых командах, хотя регулярного подхода тут не заметно. Например, скрытые файлы «скрыты» от команд DIR, DEL, COPY, ATTRIB, но их можно распечатать командой TYPE и их можно выполнить, если это выполняемые файлы. Вы можете попробовать поменять параметры файлов при помощи какого-нибудь файлового монитора (например, программы PCTOOLS), и посмотреть, как скажется такое изменение на восприятии этих файлов операционной системой.

Итак, вот набор функций, позволяющих узнать все перечисленные параметры. Функция

вид файла (имя: строка, вопрос) : лог
возвращает ответ на один из вопросов а) — е), например «вид __ файла («C:\MYDIR», КАТАЛОГ)» возвращает истину, если в корневом каталоге диска C имеется подкаталог MYDIR, и ложь, если MYDIR — это не каталог, а файл. Если, однако, в корневом каталоге объекта с именем MYDIR вообще не существует, обращение к этой функции приведет к ошибке.

Следующие три функции в комментариях не нуждаются:

размер __ файла (имя: строка) : цел

дата __ модификации (имя: строка) : дата

время __ модификации (имя: строка) : время

Конкретный вид возвращаемых даты и времени зависит от языка и мы опишем его в следующей главе. Любой из перечисленных параметров (кроме размера файла) может быть изменен с помощью таких команд:

задать __ время __ модификации (имя: vx строка, какое: vx время, удалось)

задать __ дату __ модификации (имя: vx строка, какую: vx дата, удалось)

задать __ вид __ файла (имя: строка, вопрос: vx, ответ: vx лог, удалось)

Последняя команда изменяет информацию о файле так, чтобы на заданный вопрос о файле в дальнейшем давался заданный ответ. Эту команду нельзя применять к каталогам, и ею нельзя изменить характеристики а) и е) — попытки сделать это приведут к неудаче.

Как мы уже указывали, попытка узнать или изменить информацию о несуществующем файле приведет к ошибке. Поэтому перед выполнением любой из перечисленных команд полезно убедиться, что требуемый файл действительно существует. Это можно сделать при помощи следующей функции:

файл __ существует (имя: строка) : лог

Следующие три команды дают возможность получить список файлов заданного каталога:

начать __ поиск (образец: vx строка, вид: vx, удалось)

есть __ еще __ файл : лог

следующий __ файл (имя: vx строка)

Первая команда задает образец для поиска файла. Образец должен содержать корректно записанный путь, который может включать символы «*» и «?» — как в команде DIR. На месте второго параметра может быть написано ОБЫЧНЫЙ, что означает, что надо искать все обычные файлы (т. е. не являющиеся скрытыми, системными или каталогами). Кроме того, можно указать, что помимо обычных надо искать и иные файлы, задав одну из характеристик (а), (в) или (д). Можно задать и несколько характеристик, соединив их союзом или. Например, команда

начать — поиск («C:\MYDIR*.», СКРЫТЫЙ или СИСТЕМНЫЙ, удалось)

требует от ОС подготовиться к поиску файлов, находящихся в каталоге MYDIR диска C. Поиску подлежат все обычные, все скрытые и все системные файлы. А имена подкаталогов каталога MYDIR при поиске рассматриваться не будут. Неудача может произойти, если указан некорректный путь, скажем каталога MYDIR не существует.

Следующая функция возвращает истину, если до этого был начат поиск, и есть еще нерассмотренные файлы, соответствующие заданному образцу. Во всех остальных случаях возвращается ложь.

Последняя из трех команд записывает в символьный массив длиной не менее 13 символов имя очередного найденного файла. Имя записывается без пробелов, расширение, если оно есть, отделяется точкой, завершается строка символом с кодом ноль. Команда выполняется успешно только в том случае, если функция «есть еще файл» перед этим возвратила истину.

Программа имеет возможность переименовать любой файл или каталог при помощи команды:

переименовать (старое — имя: vx строка, новое — имя: vx строка, удалось)

При переименовании обычного файла первый параметр должен указывать путь к реально существующему файлу. Второй параметр, наоборот, должен задавать путь к несуществующему пока файлу, находящемуся, однако, в реально существующем каталоге. Допускаются переименования, переносящие файл в другой каталог, но нельзя перенести файл с одного диска на другой. При переименовании каталога первый параметр должен указывать путь к реально существующему каталогу. Второй путь может отличаться от первого только последним именем, т. е. переносить подкаталог из одного каталога в другой нельзя. Отметим также, что ни в одном из параметров нельзя использовать символы «*» или «?».

И, наконец, команды создания и уничтожения элементов дерева файлов:

создать — каталог (путь: vx строка, удалось)

уничтожить — каталог (путь: vx строка, удалось)

уничтожить — файл (путь: vx строка, удалось)

Первый параметр каждой из этих команд записывается по тем же самым правилам, что и пути в командах MD, RD, DEL операционной системы — нельзя только употреблять звездочки и знаки вопроса. Как и при работе с ОС, нельзя уничтожать текущий или непустой каталог и нельзя уничтожать защищенный от изменения файл. Специальной команды создания файла нет, но это и не важно — ведь файл создается при попытке открыть его для записи, что довольно естественно: зачем создавать пустой файл, если не предполагается его ничем наполнять.

Если какая-либо из перечисленных в этом параграфе команд не удалась, можно узнать причину неудачи, изучив значение функции

номер — ошибки: цел

Вот несколько номеров ошибок, фиксируемых ОС:

1 — неверное обращение к функции ОС. Скорее всего, это означает ошибку в описании одной из процедур, реализующих перечисленные команды. Если это библиотечная процедура (см. описание соответствующего языка), это свидетельствует об ошибке в библиотеке. Во всяком случае, ошибка, скорее всего, произошла не по Вашей вине;

2 — не найден файл в заданном имени;

3 — не найден заданный путь;

4 — число открытых файлов достигло определенного предела, и нельзя открыть очередной файл, не закрыв сперва какой-нибудь другой;

5 — нарушение доступа. Причины этой ошибки могут быть довольно разнообразными. Например, попытка обратиться к обычному файлу как к каталогу или наоборот, попытка изменить защищенный от изменения файл и т. д.;

6 — сообщенный код файла не соответствует никакому открытому файлу;

16 — попытка применить команду «уничтожить каталог» к текущему каталогу. Правило «нельзя рубить сук, на котором сидишь» вполне применимо и к дереву файлов.

Обращаться к функции «номер ошибки» следует сразу после выполнения неудачной команды. Если перед этим выполнена еще какая-нибудь команда или неудачи не было, значение этой функции никакой смысловой нагрузки не несет. Приведем пример использования этой функции:

уничтожить _каталог («TMPDIR», удалось)

причина := номер _ошибки

ЕСЛИ не удалось ТО

{анализ причины ошибки}

ЕСЛИ причина = 16 ТО

напечатать («Покиньте уничтожаемый каталог!»)

и т. д.

Теперь мы расскажем немного о командах, не относящихся непосредственно к дереву файлов, но так или иначе влияющих на состояние ОС. Так, ОС рассматривает один из дисков в качестве текущего. В дереве файлов каждого диска также выделен текущий каталог. Вспомним, что во время диалога с ОС можно изменять как текущий диск, так и текущий каталог. Программа также может узнавать и менять эту информацию с помощью таких функций и команд:

текущий _диск : цел

задать _диск (какой: вх цел)

текущий _каталог: строка

задать _каталог (какой: вх строка, удалось)

Здесь диск номер ноль соответствует диску А, 1 — В, и т. д. При этом, если заданного диска не существует, команда «задать диск» не делает ничего, так что текущим остается тот же самый диск. Для смены каталога надо задать путь к нему по обычным правилам, как в команде CD операционной системы.

Еще одна команда вызывает немедленное завершение выполняемой программы:

выход (код __ завершения: вх цел)

В скобках указывается код завершения для данной программы. Именно этот код может быть проанализирован командой ERRORLEVEL, если Ваша программа запускалась внутри BAT-файла. Обычно принято завершать программу командой «выход (0)».

У читателя мог возникнуть вопрос — нельзя ли было вместо части перечисленных команд дать возможность выполнить одну единственную команду, требующую выполнить ту или иную строку, как если бы она была напечатана в режиме диалога с ОС? Можно, более того, такая команда есть. Вот она:

система (команда: вх строка, удалось)

Строка содержит текст команды ОС в той же форме в какой он должен набираться на клавиатуре в режиме диалога. Приведем несколько примеров этой команды, всякий раз сопровождая их одной из рассмотренных ранее команд, приводящих к тому же результату:

система («REN BOOK.OLD *.NEW», код)

{переименовать («BOOK.OLD», «BOOK.NEW», удалось)}

система («MD NEW», код)

{создать __ каталог («NEW», удалось)}

система («RD NEW», код)

{уничтожить __ каталог («NEW», удалось)}

система («DEL BOOK.NEW», код)

{уничтожить __ файл («BOOK.NEW», удалось)}

система («B:», код)

{задать __ диск (1, удалось)}

система («CD\», код)

{задать __ каталог («\», удалось)}

Какую из команд каждой пары предпочесть? Общее обращение к системе, видимо, нагляднее — и все же мы рекомендуем использовать в каждом конкретном случае свою процедуру. Дело, во-первых, в том, что команда «система» выполняется крайне медленно — для ее выполнения каждый раз грузится с диска файл COMMAND.COM. Кроме того, переменная «удалось» в команде «система» возвращает ложь лишь в случае невозможности передать ОС заданную строку (например, не удастся загрузить COMMAND.COM). Если же строка системе передана, возвращается истина независимо от того, как проагировала система на переданную строку.

Мы описали достаточно высокоуровневый набор для работы с внешними носителями информации — такими, как дискета или винчестерский диск. Ясно, что ОС, обращаясь к диску, переводит наши просьбы во что-то гораздо более простое, вроде «включи мотор», «передвинь записывающую головку вот сюда», «запиши вот эти байты». Такие команды может подавать и Ваша программа. Мы, однако, намеренно не станем рассказывать, как это можно сделать — слишком велика при этом цена ошибки. Ведь даже на дискете хранится масса информации: скажем, текст этой книги, а он был набран на компьютере, уместается на одной дискете. Слишком обидно было бы потерять столько информации из-за программной ошибки. Собственно, ОС в основном и создана для того, чтобы исключить такие потери.

Однако при работе в другими устройствами — экраном, клавиатурой, часами, синтезатором — цена ошибки сравнительно невысока. Поэтому мы расскажем о том, как в принципе можно управлять этими устройствами на самом примитивном уровне, т. е. без посредников вроде ОС и других служебных подпрограмм. При этом наше изложение будет вынужденно неполным, хотя мы постараемся описать минимум средств, позволяющих использовать все полезные возможности данного устройства.

5.2. ВНЕШНИЕ УСТРОЙСТВА ПЭВМ С РАЗНЫХ ТОЧЕК ЗРЕНИЯ

В предыдущей главе мы описали довольно узкий набор команд для работы с экраном и клавиатурой. Такие команды выполнимы практически на любой машине, не обязательно даже, чтобы это был ПК. Наблюдая за работой фирменных прикладных программ для ПК, мы обнаруживаем более изощренную работу с устройствами: на экране мгновенно появляются и двигаются различные картинки, можно рисовать графики и схемы и т. д. С клавиатурой порой тоже происходит нечто странное: скажем, переключение регистра воспринимается как нажатие клавиши и т. п. Кроме того, у компьютера есть ряд стандартных внешних устройств, о которых мы ничего не говорили в предыдущей главе — принтер, динамик, электронные часы. Пора рассказать и о возможностях этих устройств, и о том, как ими управлять.

Для начала нам хотелось бы обратить внимание читателя на относительность самого понятия управления. Именно: как понимать фразу «компьютер управляет внешним устройством»? Вопрос не так уж банален, ибо вот несколько ответов, каждый из которых имеет право на жизнь:

1. Это означает, что если запустить программу, то на экране начнет появляться текст

2. Это означает, что если программа подает команду «напечатать», то на экране появляется строка текста.

3. Это означает, что если операционной системе передан буфер с текстом и приказано скопировать его в файл с именем CON, то на экране появится помещенный в этот буфер текст.

4. Это значит, что если запустить стандартную программу «вшитую» в постоянное запоминающее устройство (ПЗУ) ПК, то в позиции курсора появится символ, а курсор сдвинется.

5. Это значит, что если центральный процессор выполнит определенную последовательность команд, то на экране в заданном режиме и в заданной позиции появится символ. Другие команды процессора способны поместить курсор в заданную позицию.

6. Это значит, что если на общей шине ПК появится сигнал X, то одна из вставленных в эту шину схем отреагирует на это действием Y и подаст на шину сигнал Z...

7. И т. д.

Не страшно, если не все ответы Вам сейчас понятны. Видно, и это главное, что смысл понятия «управлять устройством» относителен и зависит от того, кто управляет — пользователь, программа, ОС, набор встроенных подпрограмм, непосредственно процессор, электрический сигнал и т. п. Ощущается также своеобразная иерархия: приказ изобразить нечто на экране передается «по инстанциям» сверху вниз, пока не дойдет до уровня аппаратуры. Зачем столько уровней иерархии? Они скрывают от пользователя детали конкретного устройства. Например, для пользователя, ведущего диалог с ОС, копирование файла на экран или на принтер может быть записано единообразно — «COPY MYFILE.DOC CON» и «COPY MYFILE.DOC PRN» соответственно (попробуйте!). Другими словами, принтер и дисплей — это, с точки зрения ОС, два одинаковых устройства с разными именами, а разница, которая проявится где-то на более низком уровне обработки поданных пользователем команд, последнему не видна. Удобно, согласитесь.

Есть, правда, у такого подхода и недостаток — скрывая особенности конкретного устройства, иерархическая лестница ограничивает возможность работы с ним. Экран не принтер, его возможности значительно шире, но унификация низводит его до уровня электрической пишущей машинки. В многопользовательских системах с этим приходится мириться, так как слишком большая власть одного пользователя над каким-либо устройством может помешать другим пользователям. В ПК такой опасности нет. Пользователь имеет возможность управлять устройством на любом из перечисленных уровней иерархии. Пока наша работа «не опускалась» ниже уровня 3. В этом параграфе мы расскажем о более низкоуровневых средствах работы с устройствами, начиная с самого низкого из доступных пользователю уровня, который в нашей иерархии имеет номер 5:

работа процессора непосредственно с аппаратурой через общую шину. Но сначала несколько слов о самой этой шине и о том, что за сигналы по ней передаются.

Общая шина — это, быть может, самая поразительная особенность аппаратного устройства ПК. На первый взгляд, ничего мудреного, просто поддержан модульный принцип, позволяющий за считанные секунды вытащить из разъема сломавшуюся схему, управляющую тем или иным внешним устройством, и вставить вместо нее исправную. Такой принцип применен даже в старом ламповом телевизоре, где так несложно сменить радиолампу, вынув ее из специальной панели. Разберемся, однако, что такое классический модульный принцип, и что необычно в модулях ПК.

Работу устройства, собранного из нескольких модулей по классическому принципу, можно пояснить такой аналогией. Имеется начальник (центральный процессор), связанный по отдельным телефонным линиям с каждым из своих подчиненных (модули внешних устройств). По каждому из имеющихся в его кабинете телефонов он отдает распоряжения сообразно специальности того подчиненного, за которым закреплен этот телефон. Если заменить какого-либо подчиненного работником той же специальности, работа коллектива не нарушится, а начальник даже не узнает о подмене — вот достоинство модульного принципа. Все так. Но представим себе, что двое служащих разной специальности обменялись телефонными трубками — система тут же разладится, ни один из них не поймет передаваемых ему распоряжений. Точно так же перестанет работать телевизор, если там поменять местами пару ламп с разными наименованиями.

Поразительное отличие ПК состоит в том, что *все позиции на шине равноправны*, так что перемена местами двух вставленных в шину печатных плат никак не повлияет на работу ПК. Как же организуется такая работа? Поясним это, продолжая предыдущую аналогию. Представим, что у каждого из служащих придуманного нами учреждения, включая начальника, есть ровно по одному телефону, причем они включены параллельно, т. е. так, что всё, что говорит кто-нибудь один, слышат все. Тогда ясно, что каждый говорящий должен сперва сказать, к кому он обращается, а уж затем объяснить, с чем. Ясно, однако, что если каждый говорящий изъясняется на своем особом языке, в подобной линии связи будут страшный шум и неразбериха. Чтобы этого не случилось, язык для бесед с разными абонентами должен быть единым. Примерно так и сделано на шине ПК — по ней может курсировать лишь несколько видов сообщений. Прислушаемся к некоторым из них:

- а) «Посылаю в память по адресу М число В.»
- б) «Что находится в памяти по адресу М?»
- в) «Посылаю в порт номер Р число В.»

- г) «Что находится в порту номер Р?»
- д) «Срочно выполните программу номер N!»

Первые четыре сообщения посылает процессор, обращаясь к шине. Сообщаемые при этом числа могут принимать такие значения (мы записали их в 16-ричной системе счисления, добавив букву H, чтобы отличить их от десятичных чисел):

- M — от 0 до FFFFFH (1048575 в десятичной форме),
- P — от 0 до FFFH (4095),
- B — от 0 до FFH (255),
- N — от 0 до FFH (255).

Для начала посмотрим внимательнее на сообщение г). Что такое *порт*? В данном случае это не только не место, где швартуются суда, но даже и не специальная схема. Это чистая абстракция. Номер порта есть просто своеобразный пароль сообщения. Продолжая нашу бюрократическую аналогию, представим себе, что начальник обычно говорит в трубку: «что у нас в цехе номер 19?», «обеспечьте выполнение задания номер 37» и т. п., ожидая при этом, что кто-то ему ответит и выполнит задание, хотя и не зная, кто именно. При этом каждый подчиненный знает набор закрепленных за ним вопросов, на которые он обязан отвечать, если спросят. Начальник же в глаза не видел цеха номер 19, он знает лишь, что про этот цех можно спрашивать, а также передавать туда приказы.

Точно так же, посылая сообщение г), процессор предполагает, что найдется включенное в шину данных устройство, «ответственное» за этот порт, т. е. откликающееся при обращении процессора к порту с данным номером. Таких устройств должно быть не больше одного: если это требование нарушить, скажем вставив в две позиции на шине две идентичных схемы, то ПК, скорее всего, даже не загрузит ОС при включении. Распознав обращение к «своему» порту, устройство может выполнить какие-либо действия. Что это будут за действия — полностью зависит от конкретного устройства. Так, в случае команды г), оно может передать на шину число от 0 до 255, отвечая на заданный вопрос. Впрочем, оно может и «промолчать» — шина устроена так, что какое-то число в заданном диапазоне процессор в ответ на свой вопрос все равно получит. Аналогичные действия выполняются устройством и в ответ на сообщение в). При этом оно может воспользоваться информацией, передаваемой в порт (это, как Вы помните, число от 0 до 255), а может и не воспользоваться, а лишь прореагировать на сам факт обращения к порту.

Каким образом процессор посылает на шину такие сообщения? Просто в его системе команд есть две специальные команды, одна из которых предназначена для записи в порт, а другая для чтения из порта, т. е. программа для процессора может

включать и команды работы с портами. В нашем языке эти команды запишутся так:

прочесть __ порт (какой: вх цел, куда: вых цел)

записать __ в __ порт (какой: вх цел, что: вх цел)

Первый параметр каждой команды задает номер порта. Еще раз подчеркнем, что как чтение из порта, так и запись в порт могут изменить состояние внешних устройств. Кроме того, между операциями чтения и записи не предполагается никакой связи, т. е., если мы написали, например,

записать __ в __ порт (3D8H, 29H)

прочесть __ порт (3D8H, х),

то это вовсе не означает, что значение переменной х в результате станет равным 29H. Аналогично в паре команд

записать __ в __ порт (42H, 80H)

записать __ в __ порт (42H, 80H)

каждая команда может приводить к изменениям в работе внешних устройств, причем к разным.

За каждым внешним устройством ПК закреплен набор «своих» портов. Выполняя операции «прочесть» и «записать» с этими портами, программа может управлять соответствующими устройствами. Перечислим несколько портов ПК типа IBM, которые закреплены за стандартными внешними устройствами.

Устройство	Номера портов
Клавиатура и динамик	60H—63H
Монохромный дисплей	3B0H—3B8H
Цветной дисплей	3D0H—3D8H
Дискковод	3F0H—3F7H
Жесткий диск (IBM XT)	320H—32FH
Таймер	40H—43H
Параллельный интерфейс (принтер)	378H—37FH
Последовательный интерфейс RS232	
основной	3F8H—3FFH
дополнительный	2F8H—2FFH

Что надо писать в порты для управления устройствами? Ответ зависит от конкретного устройства. В ряде случаев необходимая последовательность манипуляций с портами может быть очень изощренной. В некоторых случаях она относительно проста. Приведем несложный пример, который Вы можете немедленно проверить на своем ПК, воспользовавшись программой DEBUG (это одна из стандартных программ ОС, содержащаяся на системной диске). Программа эта, в частности, позволяет пользователю опрашивать и изменять порты в режиме диалога.

Итак, наберем на клавиатуре DEBUG и дадим возврат каретки. В следующей строке появится знак минус — приглаше-

ние этой диалоговой системы. У Вас есть возможность давать такие команды:

i номер — порта — прочитать заданный порт,
o номер — порта, число — записать число в порт,
q — вернуться в ОС

Обозначения команд — это первые буквы английских слов input (ввести), output (вывести) и quit (закончить). Все числа должны быть 16-ричными и не выходить за пределы того диапазона, который мы указывали раньше.

Теперь можно попробовать записать какую-нибудь информацию в какой-нибудь порт. В качестве примера рассмотрим порт 61H. Часть записываемой в этот порт информации управляет клавиатурой и динамиком. Слово «часть» в данном случае означает вот что. Представим записываемое в этот порт число в двоичной системе счисления. Оно будет содержать (проверьте!) не более восьми двоичных разрядов, их еще называют битами. Мы будем записывать его так:

B7 B6 B5 B4 B3 B2 B1 B0,

где каждый бит может быть нулем или единицей, а все число образуется из двоичных разрядов по формуле

число = $B0 + 2 * B1 + 4 * B2 + 8 * B3 + \dots + 128 * B7$.

Рассматриваемый порт устроен так, что на состояние клавиатуры и динамика влияет, в частности, содержимое битов B0, B1 и B6. Остальные биты в нормальном состоянии лучше всего сделать такими: $B3 = 1$, $B2 = B4 = B5 = B7 = 0$. Ну а биты, которые мы попробуем менять, влияют на состояние упомянутых устройств так:

B6 = 1 — клавиатура реагирует на нажатие клавиш,

B1 = B2 = 1 — динамик звучит.

Попробуем заставить динамик зазвучать и умолкнуть. Вот какие комбинации двоичных цифр для этого нужны:

B7 B6 B5 B4 B3 B2 B1 B0

0 1 0 0 1 0 1 1 — динамик звучит

0 1 0 0 1 0 0 0 — динамик молчит

Переведа оба числа в 16-ричную систему, получим 4BH и 48H соответственно. Теперь подадим такие команды:

o 61, 4B

o 61, 48

После первой команды из динамика должен раздаться довольно резкий монотонный звук, который не прекратится до тех пор, пока Вы не дадите возврат каретки после набора второй команды. Заметьте, что бит B6 в записываемых в порт числах непременно должен оставаться равным единице — иначе клавиатура «оглохнет», перестав реагировать на любые нажатия. Проверьте это сами, подав команду

o 61, 8

Теперь клавиатура полностью заблокирована — она не реагирует даже на сочетание «Ctrl-Alt-Del». Придется выключить

машину и включить ее снова. Еще более эффективна команда о 61, В,

создающая эффект «невывключаемой» сирены.

Перейдем теперь к первым двум обращениям процессора к шине — командам записи и чтения памяти. Это может показаться странным, но команды работы с памятью устроены так же, как и команды работы с портами. Например, при записи в память адрес памяти и записываемые данные передаются по тем же проводам, что и при записи в порт. При этом, разумеется, один из специальных проводов шины несет информацию о том, к чему именно сейчас происходит обращение — к памяти или к порту. Поэтому обращения к памяти и порту независимы — обращение к памяти никак не влияет на устройство, управляемое через порт, и наоборот.

Такая организация с памятью означает, как и в случае портов, что память, вообще говоря, не обязана обладать своими привычными свойствами — ведь можно назначить устройство «ответственным» за те или иные адреса памяти, и тогда команды записи (и даже чтения) по этим адресам могут приводить к чему угодно. На практике, однако, большинство адресов памяти действительно используется для запоминания информации. Приведем типичную «карту памяти» ПК, подобно тому, как ранее приводили «карту портов».

Адреса	Емкость, Кбайт	Тип	Использование
0—9FFFFH	640	ОЗУ	Основная память
A0000H—AFFFFH	64	ОЗУ	Для расширенных дисплейных адаптеров
B0000H—BFFFFH	64	ОЗУ	Для стандартных дисплейных адаптеров
C0000H—CFFFFH	64	ПЗУ	Набор подпрограмм для работы с жестким диском и расширенным дисплейным адаптером
D0000H—EFFFFH	128	ПЗУ	Дополнительный набор подпрограмм
F0000H—FFFFFH	64	ПЗУ	Стандартный набор подпрограмм

Приведенная карта вовсе не означает, что вся указанная память (общим объемом в 1 Мбайт) присутствует на Вашем ПК. Так, емкость основной памяти — именно эта память используется ОС и всеми нашими программами — может меняться от 64 до 640 Кбайт в зависимости от поставки. Как правило, часть памяти, соответствующая младшим адресам (не менее 64 Кбайт), расположена на системной плате, а оставшаяся часть может быть вставлена в одну из позиций шины.

Память в диапазоне адресов A0000H—BFFFFH располагается на схемах управления устройствами, вставленных в шину. Это означает, что набор доступных в этом диапазоне адресов может не быть непрерывным. Скажем, если у Вас нет расширенного дисплейного адаптера, то и соответствующей памяти, скорее всего, просто не существует. Если на Вашем ПК имеется дисплейный адаптер (что почти всегда так и есть), то он вставлен в одну из позиций шины и может общаться с процессором через порты (их номера для стандартных адаптеров мы приводили). Кроме того, на схеме адаптера есть память, имеющая все свойства обычной памяти, т. е. там сохраняется записанная процессором информация и процессор может впоследствии считывать ее. Однако у этой памяти есть и ряд дополнительных свойств: ее содержимое влияет на содержимое экрана ПК, т. е. изменения в этой памяти приводят к изменению картинки на экране. В § 5.3 мы подробно объясним, как связаны между собой эти изменения, а пока запомним, что адреса этой адаптерной памяти заключены в названном диапазоне.

Следующие участки памяти отличаются от обычных тем, что требования процессора записать в них что-нибудь в действительности никак не влияют на хранимую там информацию. А хранится в этой памяти так называемый базовый набор подпрограмм управления устройствами (часто употребляемое в литературе название — BIOS, аббревиатура от Basic-Input—Output System, что дословно означает «базовая система ввода-вывода»). Именно этот набор поддерживает уровень 4 в иерархии, о которой мы говорили в начале этого параграфа.

Зачем нужны такие подпрограммы, если устройства все равно управляются через порты? Во-первых, затем же, зачем нужны процедуры — они экономят место в основной программе, а стандартный их набор частично избавляет от необходимости «копания в битах», которое мы предпринимали, программируя динамик. Кроме того, если наша программа построена лишь на обращениях к подпрограммам BIOS, она будет меньше зависеть от машины. Дело в том, что программирование через порты слишком ориентировано на конкретную аппаратуру — скажем, могло случиться так, что изготовитель Вашей машины применил другую схему управления клавиатурой, и тогда описанные нами действия с сиреной потерпят неудачу. Но, как правило, тот же изготовитель так «подгоняет» соответствующую подпрограмму управления клавиатурой, «зашивая» ее в память своей машины, чтобы обращение к такой подпрограмме приводило к тем же действиям, что и раньше. В результате Ваша программа, работая на иной машине, будет обращаться к тем же подпрограммам BIOS, те будут управлять иными портами, но так, что это приведет к тому же эффекту, что и на старой машине.

Подпрограммы управления стандартными устройствами (а мы только о таких и будем говорить) обычно располагаются в последнем из приведенных на «карте памяти» участков, а сам этот участок расположен на системной плате. Дополнительные нестандартные устройства могут иметь свое ПЗУ с «вшитыми» туда собственными управляющими программами. Именно для такой памяти отведены адреса D0000H—EFFFFH.

Процессор управляет памятью с помощью команд, которые мы будем записывать так:

записать __ байт (куда: адрес, что: цел)

байт __ памяти (откуда: адрес) : цел

Целое число в данном случае должно содержать не более восьми двоичных разрядов, т. е. меняться от 0 до 255. Адрес задает номер ячейки памяти емкостью в 1 байт и он может меняться от нуля до миллиона с небольшим, как мы помним, т. е. адрес — это по смыслу целое число, которое может содержать (проверьте!) до 20 двоичных разрядов. Однако здесь заключена одна из главных трудностей в программировании на ПК с процессором типа «Интел», которую не могут скрыть никакие языки высокого уровня: этот процессор не умеет работать с 20-разрядными двоичными числами, он способен обрабатывать лишь 16-разрядные. Обойдена эта трудность довольно неочевидным способом: при задании адреса нужно вместо одного 20-разрядного числа задать процессору два 16-разрядных. Одно из этих чисел называется базой, другое — смещением. Полный 20-разрядный адрес вычисляется лишь в момент обращения к шине по формуле

$$\text{АДРЕС} = 16 \times \text{БАЗА} + \text{СМЕЩЕНИЕ}$$

Умножение на 16 добирает недостающую разрядность, а смещение позволяет адресовать любой байт (а не только кратный 16). Поэтому во всех языках, в которых есть возможность непосредственно работать с адресами, адрес — это пара 16-разрядных чисел. Кроме того, как нетрудно понять, разным парам могут соответствовать одинаковые адреса, так что подумайте дважды, прежде чем пользоваться, например, операциями сравнения адресов.

Отличие памяти от портов, как мы уже уяснили, принципиальное, и заключается в принятом способе использования того и другого. Так, разные порты, даже с соседними, т. е. отличающимися на единицу адресами, как правило, используются для совершенно различных целей. Память, наоборот, чаще используется большими непрерывными участками: для хранения программы, массива данных и т. д. Поэтому в процессоре предусмотрен ряд дополнительных команд для работы с несколькими байтами памяти. Вот как мы их запишем на нашем языке:

записать __ слово (куда: адрес, что: цел)

слово __ памяти (откуда: адрес) : цел

копировать __ память (куда: адрес, откуда: адрес, сколько: цел)

Последняя команда позволяет скопировать заданное число байтов из одного места памяти в другое. Пересылаемые участки могут произвольно перекрываться в памяти. Первые две команды аналогичны командам записи и чтения байта, но только работают с двухбайтовыми словами. В связи с использованием этих команд может возникнуть такой вопрос. Пусть мы записали какое-нибудь целое число (т. е. 2 байта) по некоторому адресу M . Тогда, очевидно, «байт — памяти (M)» и «байт — памяти ($M + 1$)» — это части записанного числа. Но какие? Отвечаем: рассматриваемый процессор записывает младший байт числа по адресу M , и старший — по адресу $M + 1$, т. е. для любого адреса M справедливо такое равенство:

$$\text{слово — памяти } (M) = 256 \times \text{байт — памяти } (M + 1) + \text{байт — памяти } (M)$$

Осталось рассмотреть последнее из передаваемых по шине сообщений — сообщение «срочно выполни действие N ». Такие сообщения называются прерываниями, и предназначены они для процессора. Сигнал прерывания заставляет процессор немедленно прекратить выполняемую им программу и перейти к выполнению специальной подпрограммы, называемой подпрограммой обработки прерывания. Как правило, с такими просьбами обращаются к процессору устройства ввода, а также таймер. Прерывания от устройств ввода означают, что пришла новая информация извне, и процессор должен немедленно решить, как ею распорядиться. Таймер просто прерывает работу процессора через равные промежутки времени (при включении частота таймерных прерываний устанавливается равной 18,2 Гц). Процессор может прерывать и сам себя, что по смыслу эквивалентно вызову процедуры. Это может происходить как при выполнении некорректной операции (деление на ноль), так и «по собственному желанию» программы, для чего имеется специальная команда «возбудить прерывание номер такой-то». Все «защиты» в ПЗУ подпрограммы BIOS вызываются именно таким способом. Приведем несколько номеров подпрограмм обработки прерываний (чаще говорят просто «номеров прерываний») и устройств, за которыми закреплены эти прерывания.

Номер прерывания	Кто его создает	В каких случаях
0	Процессор	Деление на ноль
4	Процессор	Переполнение
8	Таймер	Каждые 55 мс
9	Клавиатура	Нажата или отпущена клавиша

Что делает процессор по приходу сигнала прерывания (от устройства или от самого себя)? Он переходит к выполнению подпрограммы по адресу, который хранится в таблице, расположенной в самом первом килобайте основной памяти. Таблица эта устроена так:

Адреса	Содержимое
0—1	Адрес подпрограммы для прерывания номер 0 (смещение)
2—3	Адрес подпрограммы для прерывания номер 0 (база)
4—5	Адрес подпрограммы для прерывания номер 1 (смещение)
6—7	Адрес подпрограммы для прерывания номер 1 (база)

1022—1023 Адрес подпрограммы для прерывания номер 255 (база)

В процессе начальной загрузки ПК в эту таблицу заносятся адреса подпрограмм BIOS. Однако ничто не мешает пользователю записать в эту таблицу адрес своей подпрограммы, изменив таким образом стандартный порядок обслуживания устройств. Мы не будем описывать, как именно это делается, но принципиальная возможность этого, как видим, имеется.

5.3. КЛАВИАТУРА

Описанные в предыдущей главе команды ввода с клавиатуры применимы практически на любой ЭВМ. Теперь мы расскажем о более специфических возможностях управления клавиатурой, которые реализованы на ПК. Эти особенности нужно учитывать уже при работе с описанной нами командой «получить символ». Поэтому прежде чем рассказывать про другие команды работы с клавиатурой, разберем пару, на первый взгляд, отвлеченных вопросов, имеющих, тем не менее, прямое отношение к этой команде.

Что есть символ? Давайте подумаем, что происходит при выполнении команды «получить символ». А именно, какое значение получает использованная в команде ввода переменная? Можно, кажется, ответить так: «значением переменной становится введенный символ». Тогда придется отвечать на эквивалентный вопрос — а что за символ вводится в программу при нажатии на клавишу? Опять можно дать внешне простой, а на самом деле не слишком ясный ответ: «Это значит, например, что если вы нажали на клавишу с буквой А, то будет введен символ А. А если стереть изображение буквы А с клавиши? Здравый смысл подсказывает, что ничего не изменится — каждой клавише соответствует определенный код, и когда этот код попадет в ПК, тот догадается, что введена именно буква А. Но компьютер не знает азбуки: все, на что он способен — это отличить один код от другого. Видимо, мы неявно предполагаем, что свойство этого кода быть кодом именно буквы А как-то проявится позднее. Например, если мы потребуем вывести символ с этим

кодом на экран, то на нем появится изображение именно этой буквы, а не какой-нибудь еще. Но вот незадача — у устройства, ведающего изображением символа на экране, может быть совершенно иное мнение по вопросу о том, какой код считать кодом буквы А...

Необходимость стандарта очевидна, и таким стандартом для ПК является стандарт ASCII. Он определяет, какие символы соответствуют кодам от 32 до 127. Таблица этих символов есть практически в любом руководстве или справочнике, и мы не будем ее приводить, отметив лишь, что сюда входят цифры, большие и маленькие латинские буквы и все знаки пунктуации — обычные и компьютерные. Можно гарантировать, что если Вы наберете на клавиатуре Вашего ПК любой из символов ASCII, а затем любым способом выведете их на экран, там появится как раз то, что было введено. Кроме того, на ряде зарубежных машин, поставляемых в нашу страну, имеется возможность вводить с клавиатуры и русские буквы, для чего используется специальная программа, а на клавиши рядом с изображением латинских букв наклеиваются изображения русских. Эта программа использует какую-либо функциональную клавишу (например, сочетание Alt-F10) как переключатель «русские/латинские», так что при переключении в «русский» режим нажатие алфавитных клавиш (и нескольких других — ведь русских букв больше) приводит к передаче в программу кодов, соответствующих русским символам.

Каково же это соответствие? К сожалению, здесь нет единого стандарта. Приведем стандарт, который, как нам кажется, более распространен. Это так называемый альтернативный стандарт, опубликованный в качестве проекта в [37]. В нем числам 128—159 соответствуют большие буквы, кроме буквы Ё, числам 160—175 — маленькие буквы от а до п, числам 224—241 соответствуют буквы р, с, т,..., ю, я, Ё, ё. Возможно, что эти символы могут быть отображены и на Вашем экране. Возможно также, что они закодированы в альтернативном стандарте. Возможно, но не обязательно — например, на EC1840 стандарт несколько иной.

Стандарт ASCII задает также сочетания клавиш, необходимых для набора кодов от 1 до 31. В частности, коды 1—26 получаются при нажатии Ctrl-A — Ctrl-Z, код 27 — при нажатии Esc и т. п. Однако предупреждаем: эти клавиши могут восприниматься специальным образом как при вводе, так и при выводе информации, поэтому не стоит использовать их везде, где допустимы обычные. Очевидный пример — клавиша Ctrl-C. Тем более нечего надеяться, что если вы введете символ «стрелка вверх», а потом выведете его на экран, то это приведет к соответствующему смещению курсора. Более того, такие клавиши, как стрелки, функциональные ключи, и т. п. — все клавиши, на которых не нарисовано ASCII-символа, надо об-

рабатывать специальным образом, потому что им не соответствует никакой ASCII-код. В ОС компьютера по поводу этих клавиш принято следующее решение: нажатие каждой из них воспринимается как ввод двух символов, первый из которых имеет код ноль, а второй содержит так называемый *расширенный код* введенного символа (РК). Вы легко сможете выяснить, какие коды соответствуют тем или иным символам, запустив на ПК такую программу.

ПРОГРАММА

с : симв

ПОКА ИСТИНА ПОВТОРЯТЬ

{ повторять вечно }

получить — символ (с)

перевести — строку

напечатать (код — символа (с))

КОНЕЦ ПОКА

КОНЕЦ ПРОГРАММЫ

Здесь функция «код символа» возвращает целое число, которое команда «напечатать» выводит на экран в десятичном виде, т. е. после запуска такой программы при нажатии на любую клавишу на экране с новой строки будет печататься в десятичной системе код попавшего в программу символа. Например, если после запуска такой программы нажать «а», будет выведено 65, если нажать «+» — 43 и т. д. А вот при нажатии на клавишу «стрелка вверх» на экране появятся сразу два новых числа — 0 и 72. Это значит, что тело цикла выполнилось дважды, т. е. в программу поступила пара символов с кодами 0 и 72 соответственно. Комбинации специальных клавиш с клавишами Ctrl, Shift, Alt и сочетания клавиши Alt с обычными клавишами порождает почти столько же расширенных кодов, сколько есть ASCII-кодов (РК заключены в диапазоне от 0 до 132, но не все из этих чисел использованы). Приведем несколько примеров РК.

Клавиши	Расширенные коды			
	Одиночная клавиша	+ Ctrl	+ Alt	+ Shift
F1—F10	59—68	94—103	104—113	84—93
Стрелка влево	75	115	Her	Her
Стрелка вправо	77	116	Her	Her
PgUp	73	132	Her	Her
PgDn	81	118	Her	Her
Ins	82	Her	Her	Her
Del	84	Her	Her	Her

Как видим, некоторые сочетания символов не сообщают РК команде «получить символ»: либо потому, что они дают обычный ASCII-код, либо такое сочетание вообще не приводит к вводу символа, либо используется для специальных целей. В последнем

случае мы имеем в виду клавишу Alt: в сочетании с дополнительной цифровой клавиатурой (это там же, где стрелки) она позволяет вводить символ с любым десятичным кодом. Делается это просто: надо нажать Alt, набрать на цифровой клавиатуре десятичный ASCII-код символа, который Вы хотите ввести, а затем отпустить Alt. Соответствующий символ будет введен в программу. В случае нашей программы это означает, что на экране будут появляться именно те числа, которые мы вводим при нажатой клавише Alt. Заметьте, что таким способом порождаются лишь обычные (не расширенные) коды.

Таким образом, команда «получить __ символ» дает нам возможность анализировать как обычные, так и специальные символы, что очень существенно при построении диалоговых программ. Теперь обсудим еще один важный вопрос, связанный с командой «получить __ символ».

Речь идет об *очереди введенных кодов*. В гл. 4 мы объясняли, что в момент выполнения команды «получить символ» ПК приостанавливается и ждет, пока на клавиатуре чего-нибудь не нажмут. А что произойдет, если нажать клавишу до того, как будет выполнена команда «получить символ»? Или как поведет себя описанная нами только что программа, если мы станем нажимать клавиши быстрее, чем программа успеет их обработать?

Проведем эксперимент. Наберем на клавиатуре название какой-нибудь команды или программы ОС, которая выполняется достаточно долго. Можно, например, набрать команду TREE/F. Дадим возврат каретки, команда начнет выполняться, а мы тем временем нажмем на клавиатуре какие-нибудь клавиши, скажем, Q, W, E. Дождемся, пока выполнение команды прекратится. После этого на клавиатуре снова появится приглашение ОС и набранные нами буквы. Это означает, что, выполняя команду, компьютер нашел время запомнить, что было нажато. Вы, наверное, уже догадались, что это оказалось возможным благодаря прерываниям. Действительно, при нажатии на клавиатуре любой клавиши процессор переходит к подпрограмме обработки прерывания. В результате работы этой программы соответствующий символ запоминается в специальном буфере, который находится в памяти ПК. Этот буфер называют еще *очередью*, потому что символы попадают из него в программу в том же порядке, в каком они пришли с клавиатуры, т. е. сохраняется *очередность*. Правда, есть сочетания клавиш, пропускаемые вне очереди, таковы Ctrl-Alt-Del и Ctrl-Break. Помимо очереди нажатых клавиш, ПК постоянно следит за состоянием клавиш смены регистра, т. е. помнит, нажат ли Shift (верхний регистр), включен ли режим Caps (большие буквы) и т. п.

Ясно, что чем больше в очереди стоит символов, тем большую память они занимают. Поскольку под очередь в ОС ПК отведен

фиксированный участок памяти, она не может быть слишком длинной. Мы легко поймем, как ОС решает проблему длинной очереди, немного изменив условия предыдущего эксперимента. Подадим опять команду TREE/F и сразу нажмем на какую-нибудь клавишу, скажем, Q, и будем держать ее, не отпуская. При этом клавиатура будет постоянно посылать в ПК символ Q. Через некоторое время Вы услышите характерный «писк» из динамика. Что произошло? Очередь стала слишком длинной, и теперь все последующие нажатия игнорируются, а введенные символы теряются, и каждая такая потеря сопровождается звуковым сигналом. Отпустим клавишу и дождемся окончания работы команды TREE. Вы увидите, что на экране после системного приглашения появилась строка, состоящая из букв Q. Сосчитайте, сколько букв в этой строке, и Вы узнаете максимальную длину очереди (в системе MS-DOS она составляет 15 символов).

Рассмотрим устройство очереди введенных кодов подробнее, поскольку в ней содержится больше информации, чем можно получить от команды «получить символ». Дело в том, что каждый символ в очереди кодируется двумя байтами. Один из них содержит ASCII-код символа, или ноль, если у этого символа нет ASCII-кода. Второй байт содержит расширенный код. Оба этих байта формируются программой обработки прерываний при нажатии клавиши и добавляются в очередь. Разберемся с тем, что это за коды.

Собственно, с ASCII-кодом и так все ясно, а вот свойства расширенного кода символа стоит оговорить отдельно.

Как мы уже сказали, ПК формируется для любого символа, поставленного в очередь. Команда «получить символ» игнорирует ПК, если ASCII-код символа ненулевой, потому и получается, что она иногда вводит два кода при нажатии одной клавиши, а иногда только один.

Некоторые ASCII-символы, скажем, звездочка, знаки плюс и минус, цифры, могут быть набраны на клавиатуре разными способами, т. е. в очереди введенных кодов одинаковым ASCII-кодам могут соответствовать нажатия на разные клавиши. С ПК ситуация иная: разным клавишам обязательно соответствуют разные ПК. Например, если набрать цифру 5 на основной клавиатуре, то получится ASCII-код 53 и ПК 6, а если ту же цифру набрать на дополнительной цифровой клавиатуре (включив верхний регистр либо режим NumLock), это снова даст ASCII-код 53, но с ПК, равным 76.

Случай ввода символа путем набора его десятичного кода при нажатой клавише Alt также можно распознать, проанализировав ПК, который в этом случае будет равен нулю.

ПК обычной клавиши не зависит от состояния регистров. Так, сочетания Shift-X и Ctrl-X дают тот же ПК, что и просто клавиша X. Для специальных клавиш это, как мы видели, не так.

Информация, возвращаемая командой «получить символ» — это, как правило, как раз то, что нам нужно. Но, как видим, непосредственная работа с очередью может дать больше информации, что иногда бывает полезно. Вот несколько функций, позволяющих получать информацию прямо из очереди:

есть — символы: лог

Эта функция позволяет понять, содержится ли сейчас в очереди хоть один символ, т. е. было ли нажато что-нибудь на клавиатуре. Заметьте, что эта функция, в отличие от команды «получить символ», ничего от клавиатуры не ждет, а лишь рапортует о ее нынешнем состоянии. Такая функция незаменима, например, при необходимости встроить в нашу программу возможность остановки по нажатию какой-либо клавиши.

Например, программа

ПРОГРАММА

к : цел

ПОКА не есть — символы ПОВТОРЯТЬ

перевести — строку

напечатать (к)

к := к + 1

КОНЕЦ ПОКА

КОНЕЦ ПРОГРАММЫ

будет печатать числа 1, 2 и т. д., пока на клавиатуре чего-нибудь не нажмут.

Следующие две функции возвращают информацию о первом символе очереди:

ASCII — код: цел

расш — код: цел

Эти функции тоже ничего не ожидают от клавиатуры. В частности, если очередь пуста, обе функции возвращают бессмысленные значения. Поэтому перед тем, как обращаться к этим функциям стоит проверить, есть ли символы в очереди.

Следующая команда изымает из очереди первый символ. Изъятый символ теряется:

удалить — символ

Из очереди удаляется пара байтов, соответствующая первому из имеющихся там символов. Если удалять нечего, т. е. на клавиатуре ничего не нажато, программа останавливается и ждет, пока не нажмут какую-нибудь клавишу. Как видим, эта команда похожа на команду «получить символ», хотя есть и отличия:

удаляемый символ никуда не помещается — он выбрасывается из очереди и бесследно исчезает;

любой символ — обычный и специальный — удаляется из очереди единообразно. Команда «получить символ», как мы помним, действует сложнее — специальные символы удаляются «в два приема».

Есть еще одна команда, удаляющая из очереди сразу все имеющиеся там в данный момент символы:

опустошить __ очередь

Ее можно описать и на нашем языке, пользуясь уже имеющимися функциями и командами:

ПРОЦЕДУРА опустошить __ очередь

ПОКА есть __ символы ПОВТОРЯТЬ

удалить __ символ

КОНЕЦ ПОКА

КОНЕЦ ПРОЦЕДУРЫ

Но стандартная команда работает быстрее и время ее работы не зависит от длины очереди. Как видим, эта команда, в отличие от «удалить символ», ничего не ждет от клавиатуры: если к моменту ее выполнения очередь и так окажется пустой, то ничего не делается. Если в программе перед каждой командой «получить символ» поместить команду «опустошить очередь», то программа не будет воспринимать клавиши, нажатые до того, как она об этом попросит. Иногда это удобно.

Список ASCII-кодов и расширенных кодов всех клавиш можно выяснить, выполнив такую программу:

ПРОГРАММА

ПОКА ИСТИНА ПОВТОРЯТЬ

перевести __ строку

напечатать («Нажмите любую клавишу»)

ПОКА не есть __ символы ПОВТОРЯТЬ

КОНЕЦ ПОКА

перевести __ строку

напечатать («ASCII=», ASCII __ код)

напечатать («PK=», расш __ код)

удалить __ символ

КОНЕЦ ПОКА

КОНЕЦ ПРОГРАММЫ

Заметьте, что внутренний цикл вообще не имеет тела. Это означает, что программа будет только опрашивать очередь на предмет наличия символов, ничего более не делая. Запустив такую программу, мы можем нажимать разные клавиши и смотреть, какие коды получила программа. Кодировка символов, вообще говоря, стандартная, хотя бывают и отклонения (например, на EC1840 кодировка несколько иная, чем на IBM).

Команды работы с очередью требуют некоторой осторожности. Предположим, например, что мы хотим проверить, нажата ли была только что клавиша «стрелка влево». Казалось бы, достаточно написать в программе

ЕСЛИ расш __ код=75 ТО...

и мы получим то, что хотели. Но это будет неверно, так как, если очередь пуста, то функция «расш __ код» вернет бессмысленное значение, которое может случайно оказаться равным 75. Следует, видимо, проверить, что очередь не пуста, переписав последнюю строчку так:

ЕСЛИ расш __ код=75 и есть __ символы ТО...
Но неожиданности возможны и здесь. Именно, представим себе, что в момент обращения к функции «расш __ код» очередь пустовала и случилось так, что функция вернула как раз 75. После этого сразу же была нажата какая-нибудь клавиша, скажем, «стрелка вниз», так что следующее обращение к очереди показало, что символы есть. В результате мы опять не получили желаемого. Конечно, такая ситуация крайне маловероятна, но настоящая программа не должна быть «почти правильной»! Как видим, применение описанных команд требует определенной квалификации, и мы рекомендуем начинающим не злоупотреблять ими.

Последняя функция, которую мы опишем, позволяет узнать состояния клавиш переключения регистров. Заметьте, что разные регистровые клавиши работают по-разному. Некоторые клавиши для работы в другом регистре нужно все время держать нажатыми, таковы Shift, Ctrl и Alt. Другие работают как кнопки переключатели — при одном нажатии соответствующий режим включается, при следующем выключается. Не все из перечисленных ниже переключателей регистров используются ОС, так что Вы, возможно, и не знали ранее, что нажатие этих клавиш что-то переключает. Не исключено, что какие-то из них Вам окажется удобно использовать в своей программе. Вот как выглядит функция опроса всех этих клавиш:
регистры: цел

Она возвращает целое число от 0 до 255, биты которого имеют следующие значения:

- B0=1, если нажата правая клавиша Shift,
- B1=1, если нажата левая клавиша Shift,
- B2=1, если нажата клавиша Ctrl,
- B3=1, если нажата клавиша Alt,
- B4=1, если включен режим ScrollLock,
- B5=1, если включен режим NumLock,
- B6=1, если включен режим CapsLock,
- B7=1, если включен режим Insert.

Итак, мы научились узнавать, что было набрано на клавиатуре, с точностью до конкретной клавиши. Эти возможности часто используются при разработке диалоговых систем. Именно там пользователю бывает удобно рассматривать клавиатуру не как средство для набора текста, а как управляющую панель того или иного прибора: кульмана, осциллографа, авиалайнера. Но в этом случае и экран должен имитировать не заправленный в пишущую машинку листок, а чертежную доску, осциллографическую трубку или пейзаж в иллюминаторе соответственно. Для того чтобы это обеспечить, необходимо научиться как можно более гибко управлять экраном. О возможностях управления экраном сейчас и пойдет речь.

5.4. УСТРОЙСТВО ОТОБРАЖЕНИЯ

Вряд ли будет ошибкой сказать, что половина, если не две трети привлекательности ПК заключается в возможностях визуального представления информации. Вместе с тем ПК сам по себе не слишком требователен к устройству отображения информации. Перечислим возможные конфигурации «ПК + устройство отображения».

В качестве простейшего устройства отображения информации для ПК может, в принципе, служить любой *стандартный дисплей* от большой ЭВМ, оснащенный последовательным интерфейсом RS232. Во всяком случае, ОС допускает использование такого дисплея. Мы, однако, не будем рассматривать этот вариант как потому, что при этом ПК выглядел бы вдвое (если не втрое) менее привлекательно, так и потому, что большинство прикладных систем требуют более совершенного устройства отображения и на такой вариант не рассчитаны.

Дисплейным адаптером называется устройство, которое можно вставить в шину ПК. Снаружи к адаптеру может быть подключен монитор — цветной или черно-белый. Монитор отличается от стандартного телевизора как более высоким качеством изображения, так и несколько иным способом кодирования входного сигнала. Впрочем, к дисплейному адаптеру можно подключить и телевизор, для чего, правда, может понадобиться специальное согласующее устройство. Стандартные дисплейные адаптеры обеспечивают получение на экране до 16 фиксированных цветов одновременно. Получаемые графические изображения могут иметь размер до 640×200 точек, правда, с меньшим числом цветов. Есть расширенные модели, обеспечивающие получение изображения размером до 720×350 точек при одновременном отображении 64 различных цветов. Фирмы-изготовители таких адаптеров стремятся сделать их совместимыми со стандартными. Иногда это получается.

Видеоплаты специального назначения устроены так же, как и адаптеры, только они еще мощнее: обеспечивается качество изображения до 1000×1000 точек с одновременным отображением 256 цветов. Обычно для построения изображения используется отдельный монитор. Как правило, подобные системы выпускают для специальных применений, где без высококачественного изображения нельзя обойтись, и к ним поставляется набор программного обеспечения, ориентированного на это применение. Часто такие системы комплектуются устройствами ввода изображения — телекамерой, тепловизором и т. п. Недостатком таких систем, помимо высокой их стоимости, является отсутствие стандарта, так что соединить их с пакетами общего назначения не удается.

Дисплейные адаптеры подверглись определенной стандартизации, именно на них ориентировано большинство прикладных программных систем, о них мы и поведем речь. Дисплейный адаптер представляет собой довольно сложное устройство, содержащее процессор и память. О принципе работы дисплейного адаптера мы уже говорили: он состоит в том, что процессору ПК предоставляется возможность записывать информацию в адаптерную память, за которой закреплен определенный диапазон адресов. Процессор адаптера все время читает эту память и отображает прочитанное на экране монитора. Такое «разделение труда» между процессорами — свойство столь многих отображающих систем, что мы рискуем назвать его общим. Вот что имеется в виду.

Всякая показывающая изображение система может быть, порой достаточно условно, поделена на две функционально различные части, которые мы назвали бы ЧТО и КАК. Часть ЧТО содержит информацию о формируемом изображении, а часть КАК формирует из этой информации окончательную картинку. Пример: эту страницу можно рассматривать невооруженным глазом, сквозь очки, в бинокль, вверх ногами, через калейдоскоп и т. д. При этом наблюдаемое глазом изображение может меняться до неузнаваемости, в зависимости от того, КАК мы смотрим, но все же то, на ЧТО мы смотрим, остается при этом неизменным. Такие же части можно выделить и у дисплейного адаптера.

Одна из этих частей — часть «Что» — это просто участок оперативной памяти, расположенный на схеме адаптера (этот участок принято называть видеопамятью). Другая часть — часть «Как» — специальная схема, преобразующая записанную в видеопамети информацию в картинку. Эта схема формирует видеосигнал, подаваемый на монитор, постоянно опрашивая при этом участок «Что» оперативной памяти. Процессор ПК (а значит, и наша программа) может влиять на представленную на экране картинку двумя способами. Во-первых, он может записать любую информацию в видеопаметь. Во-вторых, он имеет возможность изменить параметры схемы, которая формирует изображение: для этого у адаптера есть ряд управляемых портов, номера которых мы приводили в § 5.2. Иными словами, ПК может задать различные режимы преобразования видеопамети в картинку.

Для обеспечения совместимости ПК с разными мониторами и даже с разными адаптерами установлено несколько стандартных режимов преобразования. Как и всё в компьютере, режимы пронумерованы. Мы сейчас перечислим эти режимы, рассматривая их с точки зрения внешних отличий: какой монитор надо использовать, сколько градаций и с каким разрешением можно получить, есть ли возможность отображения графической информации

Номер режима	Вид	Число цветов или градаций яркости	Размер картинки	Монитор
0	Текст	16	40×25 символов	Чб
1	»	16	40×25 символов	Цв
2	»	16	80×25 символов	Чб
3	»	16	80×25 символов	Цв
4	Графика	4	320×200 точек	Цв
5	»	4	320×200 точек	Чб
6	»	2	640×200 точек	Чб
7	Текст	4	80×25 символов	Чб
10	Графика	4	640×200 точек	Цв
13	»	16	320×200 точек	Цв
14	»	16	640×200 точек	Цв
15	»	4	640×350 точек	Чб
16	»	64	640×350 точек	Цв

Мы можем узнать, в каком режиме работает сейчас адаптер, запросив значение функции

режим __ экрана : цел

Можно также изменить режим, подав команду

задать __ режим __ экрана (какой : вх цел)

Естественно, задаваемый режим должен допускаться стоящим на Вашем ПК адаптером, иначе программа, скорее всего, «зависнет».

Стандартный *цветной графический адаптер* (ЦГА) обеспечивает работу в режимах с 0 по 6 включительно. Мы расскажем подробно об одном символьном и одном графическом режиме ЦГА. Начнем с режима 3, который, как видно из приведенного только что списка, является текстовым. Именно этот режим обычно устанавливается сразу после включения ПК. Он позволяет изобразить в любой из 2000 позиций экрана (25 строк по 80 позиций) любой символ из некоторого стандартного набора. Есть всего 256 различных символов. Каждый символ может быть нарисован одним из 16 цветов на фоне, который может быть восьми различных цветов (черный — это тоже цвет). Символ может быть мигающим или нет. Подчеркнем, что все эти свойства индивидуальны для каждого символа, т. е. на экране могут присутствовать одновременно символы всевозможного вида. Давайте сообразим: сколькими способами можно нарисовать, например, символ А в заданной позиции экрана? Простое вычисление дает нам ответ 256. Конечно, среди этих вариантов будут и бессмысленные, вроде черного символа на черном фоне, но мы хотели понять, сколько памяти нужно, чтобы запомнить атрибут символа (так называется информация о виде данного символа на экране, включающая цвет, фон и мигание). Теперь ясно, что как на символ, так и на его атрибут требуется по 1 байту памяти. А вся картинка требует 4000 байтов, т. е. чуть меньше 4 Кбайт.

Теперь несложно будет понять, как устроена видеопамять в этом режиме. Видеопамять ЦГА занимает в ПК адреса B8000H — BBFFFFH, т. е. ее объем 16 Кбайт. Так что для построения изображения в этом режиме одновременно требуется лишь четверть всей видеопамати. Эту визуализируемую в данный момент часть видеопамати принято называть видимой страницей. Общее число страниц видеопамати, очевидно, равно четырем. Страницы имеют номера от 0 до 3 и расположены в памяти так:

Номер страницы	Адреса
0	B8000H — B8FFFFH
1	B9000H — B9FFFFH
2	BA000H — BAFFFFH
3	BB000H — BBFFFFH

Первоначально — при задании этого режима — в качестве видимой страницы выбирается страница номер 0. Содержимое остальных страниц при этом не влияет на вид экрана. Программа может узнать, какая страница является видимой, изучив значение функции

страница __ экрана: цел

Можно сделать видимой любую из страниц, подав команду задать __ страницу (какую: vx цел)

Организуя смену страниц, можно создавать на экране различные эффекты (мультипликация и т. д.). Следует, однако, сказать, что и при выводе информации прямо на экран изображение меняется, по нашим субъективным ощущениям, мгновенно. Посмотрим, как устроена каждая страница. Очень просто: каждый байт с четным адресом превращается на экране в символ, а следующий за ним байт задает атрибут этого символа. Порядок расположения преобразуемых символов на экране обычный — слева направо и сверху вниз, т. е. адрес байта, ответственного за символ в позиции C строки L, можно вычислить так (здесь и далее мы нумеруем строки и столбцы с нуля):

адрес = $2 \times (L \times 80 + C) + \text{адрес начала страницы}$

Обратите внимание, что в конце каждой страницы остается по 96 байтов, которые никогда не отображаются на экране.

Посмотрим теперь, как кодируется символ и его атрибут в видеопамати. Начнем с самих символов. Каждому записанному в четном байте числу от 0 до 255 соответствует рисунок, который будет отображаться на экране. При этом, как правило, выполнены такие условия:

- 1) символу 0 соответствует пустое место;
- 2) числам 32—127 соответствуют символы ASCII;
- 3) вид символов с номерами 128—255 несколько неустойчив

от машины к машине. Один из вариантов — это альтернативный стандарт, частично описанный нами в предыдущем параграфе. Помимо букв, этот стандарт включает также набор символов псевдографики, позволяющий рисовать рамки и т. п;

4) символам с кодами 1—31 тоже соответствуют видимые символы, среди которых есть довольно экзотические, например, нотные знаки.

На большинстве стандартных адаптеров иностранного производства набор символов фиксирован — он хранится в специальном ПЗУ на адаптерной плате и изменить его, не сменив микросхему ПЗУ, нельзя. Большинство адаптеров отечественных ПК, напротив, позволяют программировать картинки символов.

Разберемся теперь, как устроен байт атрибута символа. Для этого нам придется записать его в двоичной системе и разбить на три части:

B7	B6 B5 B4	B3 B2 B1 B0
мигание	фон	цвет

Символ будет мигать, если $B7 = 1$. Цвет символа, как видим, может быть записан четырехразрядным двоичным числом, или, что то же самое, десятичным числом от 0 до 15:

0 черный,	8 темно-серый,
1 синий,	9 ярко-синий,
2 зеленый,	10 ярко-зеленый,
3 голубой,	11 ярко-голубой,
4 красный,	12 ярко-красный,
5 розовый,	13 ярко-розовый,
6 коричневый,	14 желтый,
7 светло-серый,	15 белый.

Так же определяется и цвет фона, только он может меняться лишь до 7. Приведем несколько примеров (проверьте их сами):

Код атрибута	Вид символа
Меньше 16	Немигающий символ соответствующего цвета на черном фоне
112	Черный символ на белом фоне
190	Мигающий желтый символ на синем фоне

Теперь мы можем менять содержимое экрана командами работы с памятью, которые мы описали в § 5.2. Удобно предположить, что имеется 4 специальных массива, описанных как-то так:

ВМ0, ВМ1, ВМ2, ВМ3: массив [25, 80] пар байтов и договоримся, что каждому элементу массива соответствует символ на одной из страниц видеопамати. Чтобы обращаться отдельно к символу и атрибуту, будем каждый элемент массива при записи в программе снабжать точкой, вслед за которой

будем писать либо «с» — символ, либо «а» — атрибут, т. е. имеют право на жизнь такие обозначения:

VM0 [19, 57]. с: = 65

VM0 [19, 57]. а: = 6

копировать — память (VM0 [2, 5], VM0 [4, 8], 10)

Первая команда записывает в 57-ю позицию 19-й строки экрана символ с кодом 65 — это маленькая латинская буква а. Запомните, что атрибут этой позиции мы не меняли, так что цвет буквы а будет тот же, что и у символа, который был в этом месте экрана раньше. Вторая команда окрашивает символ в соответствующей позиции в коричневый цвет. Наконец, последняя команда пересылает пять символов (вместе с атрибутами) из четвертой строки во вторую. Как видим, использование имен массивов избавило нас от необходимости писать абсолютные адреса. В качестве примера использования описанных возможностей рассмотрим программу, которая 80 раз циклически роллирует экран вправо:

ПРОГРАММА

сдвинуто, строка: цел

цвет, буква: симв

сдвинуто: = 0

ПОКА сдвинуто < 80 ПОВТОРЯТЬ

строка: = 0

ПОКА строка < 25 ПОВТОРЯТЬ

цвет: = VM0 [строка, 79]. а

буква: = VM0 [строка, 79]. с

копировать — память (VM0 [строка, 1],

VM0 [строка, 0]. а: = цвет

VM0 [строка, 0]. с: = буква

строка: = строка + 1

КОНЕЦ ПОКА

сдвинуто: = сдвинуто + 1

КОНЕЦ ПОКА

КОНЕЦ ПРОГРАММЫ

Попробуйте перевести эту программу на один из описанных в следующей главе языков и выполнить. Вы увидите, как имеющаяся на экране картинка постепенно уйдет за правую границу экрана, а затем появится с противоположной стороны. Не исключено, правда, что это движение будет сопровождаться сильными помехами на экране в виде «снега». Связано это с несовершенством имеющегося у Вас адаптера — такие помехи возникают из-за одновременного обращения к видеопамати процессора ПК и самого адаптера. При этом неважно, обращается процессор к визуализируемой в данный момент или к невидимой части видеопамати — если уж на Вашем адаптере есть такой дефект, он будет проявляться и в том, и в другом случае. Выход один: отключать монитор на те моменты, когда видеопамать меняется, т. е. требуется еще пара команд:

выключить __ экран
включить __ экран

В большинстве стандартных адаптеров такая возможность есть, так что умелое использование этих команд может устранить особенно неприятные явления на мониторе, сменив их менее неприятными: скажем, «снегопад» на экране сменится мерцанием.

Еще одна часть схемы адаптера занимается изображением на экране курсора. Эта часть никак не связана с видеопамятью — изображение курсора и текст смешиваются лишь на экране. Менять позицию курсора можно с помощью команды

поз __ курсор (строка: vx цел, столбец: vx цел)

где координаты курсора могут меняться от 0 до 24 по строкам и от 0 до 79 по столбцам. Впрочем, можно отправить курсор и дальше, за пределы видимой части страницы, если мы хотим, чтобы его не было видно. Команда «поз __ курсор (25, 0)» вполне подойдет для этого. Выяснить, где сейчас находится курсор, можно при помощи двух функций:

строка __ курсора: цел

столбец __ курсора: цел

Рассмотрим теперь режим 4. В этом режиме на экране изображается 200 строк точек, по 320 точек в каждой строке. При этом одновременно на экране могут быть изображены точки четырех различных цветов. Нетрудно подсчитать, что для изображения точек такой картинки потребуется одновременно все 16 Кбайт видеопамати ЦГА (точнее, 16000 байт). Как же строится соответствие между адресами видеопамати и расположением точек на экране в этом случае?

Видеопамать разбивается на две половины. Одна из них (адреса B8000H — B9FFFH) хранит цвета всех точек, находящихся в четных строках (строках 0, 2, ..., 198). Другая половина (адреса BA000H — BBFFFH) отвечает за нечетные строки изображения. Сделано это, видимо, по техническим причинам. Во всяком случае, нам с Вами остается лишь смириться с такой организацией памяти. Внутри каждой из половин каждый из байтов отвечает за цвета четырех соседних точек. Например, биты B7 и B6 байта, находящегося по адресу B8000H, определяют цвет левой верхней точки экрана. Биты B1 и B0 байта BBF3FH отвечают за цвет правой нижней точки. Заметьте, что в конце каждой половины видеопамати опять остается несколько невидимых байтов. Можно построить общую формулу вычисления адреса видеопамати, ответственного за цвет точки, находящейся в строке L и столбце C:

$$\text{адрес байта} = 80 \times [L/2] + [C/4] + \begin{cases} \text{B8000H, если } L \text{ чётно} \\ \text{BA000H, если } L \text{ нечётно} \end{cases}$$
$$\text{адрес младшего бита} = 6 - 2 \times (\text{остаток от деления } C \text{ на } 4)$$

Здесь квадратные скобки означают операцию взятия целой части. Как видим, понять, какую область памяти следует изменить, чтобы покрасить заданную точку в заданный цвет, не просто. К счастью, ПК снабжен стандартной программой, облегчающей эти действия. Мы будем называть эту программу при помощи такой команды:

покрасить __ точку (строка: vx цел, столбец: vx цел, цвет: vx цел)

Здесь «цвет» — это число от 0 до 3. Можно также узнать цвет текущей точки с помощью функции

цвет __ точки (строка: цел, столбец: цел): цел

Цвет каждой точки может меняться от 0 до 3, что и означает возможность рисовать на экране одновременно четырьмя различными цветами. Но какие это цвета? Это зависит уже от устройства преобразования памяти в картинку и, соответственно, числа — в цвет (т. е. от части КАК адаптера — не забыли нашу классификацию?). Сразу после включения режима 4 устанавливается такое соответствие:

Номер цвета	Цвет
0	Черный
1	Голубой
2	Розовый
3	Белый

Казалось бы, надо дать пользователю возможность менять это соответствие произвольным образом, задавая цвет, соответствующий каждому числу, тем более что монитор способен изображать по крайней мере 16 различных цветов — вспомните режим 3. Для цвета номер 0 так и сделано — Вы можете заставить адаптер раскрашивать точки, со значением 0 в 16 различных цветов. А вот с остальными значениями так почему-то поступить нельзя — возможны лишь два закона преобразования этих чисел в тройку цветов. Вот второй вариант преобразования:

Номер цвета	Цвет
1	Зеленый
2	Красный
3	Коричневый

Эти два варианта называют палитрами (с номерами 0 и 1 соответственно). Итак, в нашем распоряжении такие команды:

задать __ цвет 0 (какой: vx цел)

задать __ палитру (0)

задать __ палитру (1)

Приведем пример программы, которая рисует на экране график функции $V=X^2$ в подходящем масштабе. График рисуется красными точками на синем фоне.

ПРОГРАММА

X : цел

здать _ режим (4)

здать _ палитру (1)

здать _ цвет 0 (2)

X=0

ПОКА X < 200 ПОВТОРЯТЬ

покрасить _ точку (X, (X * X) / 200, 2)

X := X + 1

КОНЕЦ ПОКА

КОНЕЦ ПРОГРАММЫ

Переведем такую программу на компьютерный язык и выполним ее. Вы увидите, что график рисуется вовсе не мгновенно. Если вы серьезно собираетесь пользоваться графикой, то команды создания рисунка путем окрашивания каждой отдельной точки не подойдут. В ряде языков есть специальные команды для построения отрезков и кривых второго порядка. К другим языкам поставляется набор процедур, обеспечивающих такие возможности и написанных прямо в кодах машины. Это позволяет строить несложные графические картинки достаточно быстро. Если же Вы стремитесь к созданию быстро меняющихся графических картинок достаточно сложной структуры, то без языков низкого уровня Вам не обойтись. Мы не будем рассматривать нестандартные графические программы в этой книге.

Рассмотрим другие графические режимы ЦГА. Режим 1 отличается от режима 3 вдвое меньшим количеством символов на экране, но зато вдвое большим количеством страниц (от 0 до 7). Каждая страница, как и в режиме 3, расположена в памяти, начиная с адреса, кратного 2048, а в конце страницы остается 48 «невидимых» байтов. Этот режим применяется при выводе на бытовой телевизор, который не может обеспечить качественное отображение 80 символов в строке. Режимы 0 и 2 при использовании цветного монитора ничем не отличаются от режимов 1 и 3, но они ориентированы на черно-белый монитор. То же относится к режиму 5.

Режим 6 отличается от 4-го тем, что в каждой строке вместо 320 точек имеется 640. Достигается это тем, что каждая точка кодируется не двумя, а одним битом видеопамати. Из-за этого число цветов сокращается до двух — черного и белого. Команды задания цвета 0 и палитры могут влиять на фактически отображаемые в этом режиме цвета, но твердого стандарта на этот счет нет. Записывать и считывать точки можно теми же командами, что и в режиме 4, но с учетом сузившегося диапазона цветов.

Монохромный дисплейный адаптер (МДА) поддерживает лишь режим номер 7, аналогичный режиму 3. Перечислим отличия этого адаптера от цветного, помимо самого очевидного:

у МДА меньше видеопамять — всего 4 Кбайта, и расположена она, начиная с адреса В0000Н, т. е. команды переключения страниц недопустимы;

по-иному трактуется атрибут символа. Вот список допустимых атрибутов:

- 0 — невидимый символ,
- 7 — белый символ на черном фоне,
- 15 — то же, но ярче,
- 112 — черный символ на белом фоне,
- 120 — то же, но ярче,
- 1 — белый символ на черном фоне, подчеркнутый снизу одной чертой,
- 9 — то же, но ярче.

Мигание может быть задано тем же способом, что и раньше.

Расширенный графический адаптер обычно поддерживает большинство или даже все перечисленные в начале этого параграфа режимы. Приведем основные отличия такого адаптера от стандартных:

- возможность программировать шрифт в текстовом режиме;
- наличие нескольких страниц в ряде графических режимов;
- возможность произвольно задавать соответствие между числами и цветами в любом из режимов.

Обычно такой адаптер поддерживает все возможности стандартных. В частности, мы можем задать любой графический режим командой «задать режим» и записывать точки всех допустимых цветов на экран командой «покрасить точку».

Несколько слов об изображении символов в графических режимах. Единственный способ нарисовать символ в графическом режиме — это честно покрасить все точки, из которых он состоит, в нужный цвет. Программы, которые делают это, есть в ПЗУ ПК, так что нет надобности изобретать изобретенное. В частности, команда «напечатать» работает в любом из перечисленных режимов. При этом печать всегда начинается с позиции курсора, хотя в графических режимах он и не виден, т. е., позиционируя курсор, Вы можете менять место печати. Символы в большинстве графических режимов рисуются в растре 8×8 , так что на экране умещается по-прежнему 25 строк. Нетрудно также подсчитать, что при работе в режиме 320×200 в каждой строке умещается 40 символов, а в режиме 640×200 — 80.

Надеемся, что приведенных выше команд Вам окажется достаточно для того, чтобы построить график на экране, и снабдить его пояснительной надписью. Заметим, что мы описали не все команды управления экраном, но зато принципиальные

возможности стандартных адаптеров описаны полностью. Так что, если у Вас имеется адаптер ЦГА, не надейтесь раскрасить ваш график в 5 цветов — никакое программирование тут не поможет.

5.5. ДРУГИЕ ВНЕШНИЕ УСТРОЙСТВА ПЭВМ

Мы расскажем еще о трех традиционных для ПК устройствах: о принтере, встроенных часах (таймере) и звуковом генераторе. Необходимость программировать эти устройства возникает куда реже, чем в случае видеомонитора и клавиатуры, однако представлять себе их возможности может быть очень полезно, тем более, что программировать их гораздо проще, чем дисплейный адаптер.

Принтером называют устройство, печатающее тексты или графики на бумаге под управлением компьютера. Рассмотрим три возможности вывода информации на принтер.

Первая возможность — в любой момент работы на компьютере, скажем, в тот момент, когда на экране появился столь ожидаемый Вами результат, Вы можете распечатать его, нажав на специальную клавишу (на иностранных компьютерах эта клавиша обозначается `PrtScr` — от английского `Print Screen`, что означает «напечатать экран»). При этом срабатывает специальная программа, записанная в ПЗУ. Эта программа рассчитана на печать текстовой информации с экрана в одном из текстовых режимов. Для распечатки графических изображений к этой программе имеется «довесок», поставляемый на дискете вместе с ОС: программа `GRAPHICS.COM`. Эту программу следует загрузить всего один раз перед началом распечатывания графической информации. Распечатка графика с помощью этой программы, однако, дается не даром: программа `GRAPHICS` занимает часть оперативной памяти, которая не освободится, пока не перезагрузят ОС.

Вторая возможность работы с принтером — печать текстов путем копирования их в файл со специальным именем `PRN`. Это можно сделать и в режиме диалога с ОС, например, дав команду

`COPY MYFILE.TXT PRN`

В программе для печати текста необходимо просто записать информацию в этот файл одной из рассмотренных команд. Не пытайтесь только закрывать файл `PRN` — он всегда открыт и должен таким оставаться.

Наконец, третья возможность — обращение к программам BIOS, предназначенным для управления принтером. Вот какие команды допускаются в этом случае:

принтер __ начать (какой : вх цел)

послать __ на __ принтер (какой : вх цел, что : вх симв, ответ : вых байт)

состояние __принтера (какого : вх цел) : байт

Первым параметром каждой команды является номер принтера, к которому она обращена. К ПК можно подключить до четырех принтеров, их номера будут меняться от 0 до 3.

Первая команда инициализирует схемы, ведающие общением с принтером. Вторая — посылает на принтер заданный байт и выслушивает ответ принтера — тоже байт. Последняя команда возвращает ответ принтера, ничего ему не посылая. Различные биты ответа сигнализируют о текущем состоянии принтера или о его реакции на последнюю команду. Смысл этих битов такой:

B4=1, если принтер подключен к ПК,

B7=1, если принтер готов к выполнению команды,

B6=1, если принтер успешно выполнил предыдущую команду,

B5=1, если в принтер не заправлена бумага,

B3=1, если зафиксирована ошибка ввода/вывода,

B0=1, если принтер за достаточно большое время не смог выполнить переданную ему команду. Процедура передачи байта ждет ответа принтера некоторое время (достаточное, скажем, для прогона страницы). Если за это время ответ от принтера не поступил, устанавливается данный бит.

Остальные биты не используются. Обычный ответ принтера при нормальной работе — двоичное число 11010XX0.

Использование описанных трех команд полезно в случае выполнения каких-либо специфических операций, например загрузки шрифта, изменения режима печати и т. п. В обычном режиме распечатки текста вполне хватает возможностей ОС.

Аппаратное устройство *таймера* несложно — это счетчик импульсов кварцевого генератора, периодически вызывающий прерывание процессора. Программа обработки прерываний таймера занимается тем, что прибавляет к хранимому системой текущему времени промежуток, равный времени между последовательными таймерными прерываниями. Именно таким образом ОС узнает текущее время. Стандартные часы ОС не запоминают время после выключения компьютера, поэтому после включения машины обычно начинается диалог со словами «Введите текущее время» и т. д. Впрочем, существуют и часы с батарейным питанием и специальные программы, передающие их время часам ОС. Форма программ управления временем сильно зависит от языка программирования и мы расскажем о них в следующей главе.

В стандартном ПК существует две возможности управления звуковым генератором. Во-первых, можно то подключать динамик к источнику постоянного напряжения, то отключать его. Если эту операцию производить достаточно быстро, то из динамика раздастся звук. Вот необходимые для этого процедуры:

включить __ динамик

выключить __ динамик

Существует также возможность переключить источник на переменное напряжение любой звуковой частоты. Для этого предусмотрены две следующие команды:

включить __ колебания

выключить __ колебания

Задать период колебаний можно следующей командой:

задать __ период (какой : вх цел)

где длина периода задается в периодах задающего генератора (выполняющего ту же роль, что и маятника в обычных часах). Обычно один период этого генератора равен 210 нс.

Глава 6. ЯЗЫКИ ПРОГРАММИРОВАНИЯ ПЕРСОНАЛЬНОЙ ЭВМ

Как было бы приятно, если бы компьютер понимал тот язык, на котором мы писали программы в предыдущих главах! Невредно, однако, немного помечтать, и представить себе, что так оно и есть. Как могла бы выглядеть работа с ПК в этом случае?

Ясно, что ввести программу в компьютер все равно придется. Продолжая мечтать, можно представить себе разные способы сделать это, начиная с ввода текста голосом и кончая передачей биотоков мозга прямо в оперативную память, однако цель одна и та же — надо сделать так, чтобы текст программы стал доступен компьютеру. Читатель вправе упрекнуть авторов в недостатке воображения, но мы позволим себе прогноз: пока не будет придумано лучшего представления программы, чем представление в виде текста, основным средством ввода программы в машину останется экранный редактор или сходная программа, моделирующая функции пишущей машинки либо карандаша с листом бумаги.

Во всяком случае приемлемой и в то же время реалистичной представляется такая схема работы: Вы набираете текст программы, пользуясь командами экранного редактирования, затем, нажав специальную клавишу «выполни», заставляете ПК исполнить только что введенную программу. Разумеется, компьютер должен тактично указывать Вам на ошибки, если таковые встретятся (а кто не ошибается!). Ясно также, что должна быть обеспечена возможность хранения программ, возможность редактирования ранее введенных программ, возможность запустить на выполнение любую из хранимых программ.

Вернувшись к реальности, мы обнаружим, что она вовсе не безрадостна. Системы, правда, рассчитаны на иные языки,

но в остальном допускающие все те возможности, о которых мы тут только что мечтали, на самом деле существуют на ПК. Вспомните хотя бы примеры, которые мы уже приводили (см. § 1, 4). Перечислим основные способы выполнения пользовательских программ на ПК; о некоторых из них мы уже упоминали:

1. Компиляция. Представим себе, что на диске ПК возник выполняемый файл (в рассматриваемой нами ОС все такие файлы имеют расширение EXE или COM), в результате запуска которого компьютер сделает все то, что полагается сделать согласно Вашей программе. Тем самым задача была бы решена. Существуют программы, умеющие прочесть текстовый файл с программой пользователя и сделать эквивалентный ему выполняемый файл. Такие программы называют компиляторами, а процесс перевода программы из текстового представления в представление в виде файла с выполняемой программой — компиляцией.

2. Интерпретация. Одним из примеров интерпретирующей системы является, как это ни странно, сама ОС. Действительно, она имеет свой входной язык, т. е. набор допустимых команд и способов их употребления. На языке ОС с помощью любого текстового редактора можно писать программы и хранить их в файлах с расширением BAT. Достаточно набрать имя такого файла — и находящаяся в нем программа станет выполняться. Если по мере выполнения встретится некорректная команда — ОС сообщит об этом и т. д.

Конечно, язык ОС — это специализированный язык, основное назначение которого — обеспечить работу с файлами, как с неделимыми единицами информации. Но легко представить себе, что ОС допускает помимо стандартных команд все те команды и конструкции, что мы использовали в наших программах. При этом никакие выполняемые файлы, кроме файлов типа BAT, содержащих тексты программ, не нужны — ведь каждая команда в такой программе может быть сразу выполнена ОС, так что не требуется никакого перевода. На самом деле, задача перевода осталась — мы лишь загнали ее внутрь ОС, которая должна правильно интерпретировать все новые команды и программные конструкции. Ясно, что это усложнит ОС. Более обидно, что это еще и увеличит время выполнения программ: скажем, если тело какого-нибудь цикла выполняется сто раз, это значит, что система сто раз должна будет проинтерпретировать это тело, всякий раз переводя его в какие-то более простые и понятные аппаратуре команды.

Итак, программные системы, реализующие режим интерпретации, существуют и называются интерпретаторами. Запустив программу-интерпретатор, Вы как бы попадаете внутрь описанной только что расширенной ОС и можете работать на одном из языков. В некоторых ПК сама ОС одновременно пред-

ставляет собой систему программирования на каком-либо одном языке.

У интерпретаторов два основных недостатка. Один из них — низкая скорость работы интерпретируемых программ. Опыт показывает, что проигрыш в скорости по сравнению с откомпилированной программой составляет по меньшей мере порядок. Другой, не столь принципиальный недостаток, состоит в том, что интерпретируемым языком, как правило, является язык Бейсик, разговор о недостатках которого заслуживал бы отдельного параграфа.

Есть у интерпретатора и преимущество перед компилятором, причем фундаментальное: программа пользователя имеет только *одно представление* — в виде текста. В компиляторе одна и та же программа имеет по меньшей мере два представления — в виде текста и в виде выполняемого файла, причем второе представление является для пользователя «черным ящиком». Все сложности работы с компиляторами — сама необходимость компиляции, большие усилия на поиск и устранение ошибок и т. д. — являются результатом этого недостатка. Его можно «закамуфлировать», но нельзя устранить полностью.

3. Соединение редактирования и компиляции. Обычно входной информацией для программы-компилятора является файл с исходным текстом программы, который мы хотим превратить в выполняемый файл. Откуда и как возник этот текст, для компилятора несущественно. Но представим себе, что у компилятора есть свой персональный текстовый редактор, а точнее, компилятор и редактор составляют единый программный комплекс. Тогда пользователь имел бы возможность прямо из редактора потребовать выполнить написанную программу. Более того, при наличии ошибки компилятор может не печатать сообщения, а прямо в тексте программы указать место, где начинается ошибочный текст, например поставив туда курсор. Все это очень ускоряет процесс поиска и исправления ошибок. Развивая эту идею, можно доверить компилятору не только диагностику, но и профилактику ошибок, спроектировав текстовый редактор так, чтобы он просто не позволял пользователю набрать ошибочную программу. Среди программного обеспечения ПК существует ряд тандемов редактор-компилятор, о двух из которых мы расскажем в этой главе.

К недостаткам редакторов-компиляторов следует отнести сложность создания больших программных проектов, требующих соединения нескольких программ и редактирования связей между ними. Не исключено, правда, что эти проблемы разрешимы [30].

Итак, наши программы переводимы на компьютерный язык, и эта глава будет полностью посвящена рассказу о простейших правилах такого перевода для нескольких наиболее распро-

раненных языков. Подчеркнем — нашей целью является не рассказ о том или ином языке программирования, и даже не его обзор. Каждый параграф, посвященный одному из языков программирования ПК, следует рассматривать как краткий словарь-разговорник по соответствующему языку, т. е. пособие, полезное лишь при практической работе на компьютере с данным языком на начальном этапе обучения. Параграфы, посвященные языкам Паскаль, Си и Бейсик, построены по единой схеме: они посвящены в основном переводу построенных в гл. 4 программ на соответствующий язык и рассчитаны на человека, впервые пытающегося заставить компьютер выполнить придуманную им программу. Заключительные части каждого параграфа посвящены описанию возможностей управления внешними устройствами при работе на данном языке. Они рассчитаны на более опытного пользователя.

Какой язык выбрать? Мы затрудняемся дать готовый ответ и даже правила выбора. По нашему мнению, рассматриваемые далее языки примерно равномощны, так что критерий выбора скорее всего будет для Вас каким-то внешним: наличие компилятора, подробного справочного пособия, помощь опытного программиста, знающего один из языков, покупка нужного Вам пакета программ, ориентированного на данный язык, и т. д. Когда же Вы, наконец, определитесь окончательно, приглашаем снова взять в руки эту книжку, включить компьютер, и да сопутствует Вам успех.

6.1. ЯЗЫК ПАСКАЛЬ

Самый лучший способ обучиться иностранному языку — как можно скорее начать говорить на нем, пусть поначалу неуверенно и с ошибками. Очень важно в этом случае наличие хорошо владеющего языком собеседника, который, во-первых, может показать, как надо правильно произносить ту или иную фразу, и во-вторых, поправит Вас, если Вы ошибетесь. Нам кажется, что и при изучении языка программирования, в частности Паскаля, такой способ подойдет. При этом в качестве собеседника, приводящего примеры «правильной речи», может на первых порах, как мы надеемся, выступить эта книга, а в качестве контролера ошибок — ПК, снабженный компилятором с Паскаля.

Начать, видимо, стоит с примера, ибо компилятор не сможет Вас понять, если Вы еще не знаете ни слова. Итак, переведем на Паскаль программу 1, построенную нами в гл. 4. Для сравнения вначале дан оригинал.

ПРОГРАММА 1

напечатать («Программа 1 выполнена!»)
КОНЕЦ ПРОГРАММЫ

```
program PROG1 (input, output);  
begin  
    write ('Программа 1 выполнена!')  
end.
```

Соответствие между приведенными текстами устанавливается без труда. Отметим несколько особенностей:

программа на Паскале должна иметь имя, которое мы придумываем сами. Все остальные слова — `program`, `input`, `output` и даже точка с запятой — ключевые и должны быть воспроизведены в любой из паскалевских программ буквально;

кроме заголовка, перед началом программы должно еще быть ключевое слово `begin`, что означает «начало»;

`write` — это перевод на английский слова «записать» и дословный перевод команды «напечатать». Единственная разница — печатаемая строка заключается не в двойные, а в одинарные кавычки;

в конце паскалевской программы после ключевого слова `end` (конец) ставится точка. Не забывайте оканчивать точкой все Ваши программы.

Как выполнить эту программу? Мы опишем работу с двумя наиболее популярными Паскаль-системами на ПК. Если ни одной из них на Вашем ПК нет, то придется либо приобрести соответствующую систему, либо самостоятельно изучить руководство к тому компилятору, который у Вас имеется.

Редактор-компилятор Турбо-Паскаль фирмы Borland. Обычный комплект поставки этой системы состоит из двух дискет, на которых размещен как сам компилятор, так и многочисленные примеры паскалевских программ. С примерами Вы можете поступить, как Вам заблагорассудится, а вот компилятор стоит скопировать на отдельную дискету либо на жесткий диск в отдельный каталог (назовем его, скажем, `TPASCAL`). Скопировать нужно такие файлы: `TURBO.COM`, `TINST.COM`, `TURBO.MSG`, `TINST.MSG`.

Эти файлы занимают на дискете меньше половины объема, так что там еще останется место для Ваших программ. Теперь достаточно набрать на клавиатуре `TURBO`, и после недолгой загрузки Вы увидите картинку, изображенную на рис. 6.1. Вы, должно быть, уже догадались, что система ждет от Вас ответа на вопрос. Смысл его такой: хотите ли Вы получать осмысленные сообщения об ошибках в Вашей программе, если они паче чаяния произойдут? На всякий случай ответим нажатием клавиши `Y`, что означает «да». Тогда к загруженной в память программе добавится еще и файл `TURBO.MSG`, который

как раз и содержит тексты сообщений о возможных ошибках, и картинка на экране примет вид, показанный на рис. 6.2. И здесь все просто. Перед Вами небольшое меню. Ярче выделены символы, которые можно ввести, чтобы выполнилось соответствующее действие. Например, нажав E (edit означает «редактировать»), мы попадем внутрь текстового редактора.

```

-----
TURBO Pascal system                      Version 3.01A
                                           PC-DOSE

Copyright (C) 1983,84,85                BORLAND Inc.
-----

Color display 80x25

Include error messages (Y/N)?

```

Рис. 6.1. Экран ПЭВМ после запуска Паскаль-компилятора

```

Logged drive: C
Active directory: \

Work file:
Main file:

Edit    Compile  Run    Save
Dir     Quit    compiler Options'

Text:      0 bytes
Free: 29224 bytes

>

```

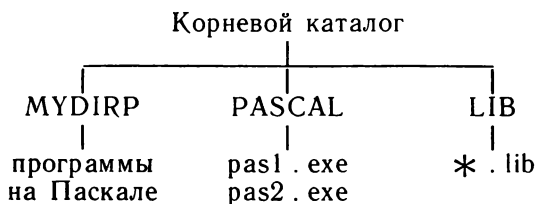
Рис. 6.2. Экран ПЭВМ при работе с Паскаль-компилятором

Сделаем это. Видно, что ни в какой редактор мы не попали, а только на экране появилась надпись «Work file name» («введите имя рабочего файла»). Пусть файл, который мы будем редактировать, называется, например, PROG1. Наберем это имя и дадим возврат каретки. Мы оказались внутри текстового редактора. Заметим, что к нашему файлу редактор добавил расширение PAS, подразумевая, что Вы наберете программу на Паскале. Сделайте это, пользуясь обычными редактирующими средствами, аналогичными тем, что есть в редакторе WORDSTAR или в редакторе SideKick. В частности, можно пользоваться стрелками, клавишами Del, «Забой», переключать режим замены/вставки клавишей Ins, уничтожать строку клавишей Ctrl-Y и т. п.

Набрав текст первой из наших программ, покиньте редактор, нажав Ctrl-K, а затем D. Вы опять увидите знак «>» — приглашение системы Турбо-Паскаль, но не увидите основного меню. Чтобы оно появилось вновь, дайте еще раз возврат каретки. Теперь достаточно нажать R — это первая буква в слове RUN, что в данном случае означает «выполнить» — и после недолгой компиляции, настолько недолгой, что Вы, скорее всего, и не успеете заметить, на экране появится текст: «Программа 1 выполнена!». Естественно, это произойдет в самом благоприятном случае — если в программе нет грамматических ошибок. Если ошибки найдутся, компилятор сообщит нечто вроде «найжены ошибки, нажмите Escape». Послушаемся его совета и нажмем на предложенную клавишу. Мы снова окажемся в текстовом редакторе, причем курсор будет поставлен в то место, дойдя до которого, компилятор понял, что программа ошибочна. Реальная ошибка может, разумеется, находиться и несколько выше по тексту, но довольно часто она расположена именно там, где указывает курсор. Исправьте ошибку и повторите процесс, и так до тех пор, пока не добьетесь выполнения программы.

С заработавшей программой можно поступить двояко. Во-первых, Вы можете записать ее исходный текст на диск, нажав S в основном меню. Вторая возможность — сохранить выполняемую программу на диске в виде файла с расширением COM. Для этого следует нажать O, войдя в подменю, задающее, куда следует поместить откомпилированную программу, затем нажать B, выбрав тем самым вариант «сохранить в файле типа COM», потом выйти из этого подменю, нажав Q, и, наконец, откомпилировать файл, нажав C в основном меню. Если после этого Вы покинете редактор-компилятор, то окажется, что в текущем каталоге появился файл PROG1.COM, который можно выполнить, просто набрав его имя в режиме диалога с ОС и дав возврат каретки. Как видим, в качестве контролера правильности написанных программ редактор-компилятор довольно удобен.

Компилятор *Microsoft* Паскаль. Этот компилятор занимает две дискеты, однако на самом деле нам потребуются не все файлы, имеющиеся там. При работе с этим компилятором на жестком диске удобно создать такую структуру:



В подкаталог PASCAL требуется скопировать указанные файлы, в подкаталог LIB — все файлы с расширением LIB. Кроме того, в файл AUTOEXEC.BAT придется добавить такие команды:

```
SET LIB=C:\LIB
```

```
PATH=то, что уже есть; C:\PASCAL
```

Не забудьте перезагрузить ОС после изменения файла AUTOEXEC — иначе сделанные изменения никак не проявятся.

Достроив файловую систему таким образом, перейдем в свс каталог, т. е. в тот, в котором мы намерены хранить свс программы на Паскале. Стоит завести отдельный каталог для этой цели. Пусть он называется, например, MYDIRP. Всю дальнейшую работу мы будем выполнять, находясь в этом каталоге. А начать эту работу надо с ввода программы в ПК при помощи любого из доступных Вам текстовых редакторов. Пусть файл с нашей программой называется PROG1.PAS. Набрав программу и вернувшись из текстового редактора в ОС, проверим на всякий случай, действительно ли в нашем рабочем каталоге появился файл PROG1.PAS, подав команду DIR. Если все в порядке, начнем компиляцию, подав такую команду:

```
pas1 prog1;
```

Понимать это следует так:

pas1 — откомпилируй, т. е. переведи с языка Паскаль на промежуточный язык...

prog1 — ...программу, находящуюся в файле prog1.pas (поскольку мы не указали расширение, компилятор добавит традиционное расширение .pas)...

« ; » — ...и не задавай лишних вопросов.

Если все вышеописанное было проделано безошибочно, то компилятор выведет на дисплей свой «товарный знак»:

Microsoft Pascal Compiler

Version 3.30 March 1985

(C) Copyright Microsoft Corp 1981, 1982, 1983, 1984, 1985

затем ПК «подумает» с десяток секунд и сообщит:

Pass One No Errors Detected.

(первый проход. Ошибок не обнаружено.)

а затем высветит приглашение ОС. Это означает, что часть компиляции прошла успешно, и программа переведена на промежуточный язык, уже достаточно близкий к машинному. Если же Вы сделаете что-то не так, как мы объясняли, на экране могут появиться и другие сообщения. Вот, например, какие сообщения возможны после подачи команды компиляции:

Bad command or file name

(здесь: не нашел такого файла)

Это сообщение Вам, видимо, знакомо. Возможно, Вы неправильно набрали команду, например вместо PAS1 набрали

просто PAS и т. п. Если команда набрана без ошибок, а это сообщение все равно появляется, это означает, что Вы либо забыли скопировать на диск (или скопировали не туда) файл `pas1 . exe`, либо не указали путь к каталогу PASCAL командой PATH в файле AUTOEXEC. Набрав команды PATH (без параметров и знака равенства), а также DIR\PASCAL, Вы быстро разберетесь, в чем дело. Если эта трудность преодолена и на экране выводится «товарный знак» компилятора — значит программа-компилятор заработала. От нее, однако, также могут поступать разные неприятные сообщения.

Приведем несколько примеров сообщений, которые могут появиться уже после начала работы компилятора:

```
Object filename [prog1 . OBJ]:
```

(куда поместить результат компиляции [может быть, в файл `prog1 . obj`]? :)

Это не сообщение о какой-то ошибке, просто компилятор задает Вам вопрос. Видимо, набирая команду компиляции, мы забыли про точку с запятой — приказ компилятору не задавать лишних (а тем более непонятных) вопросов. И вот результат — компилятор начал о чем-то нас спрашивать. У Вас есть две возможности. Во-первых, можно прямо сейчас все-таки ввести точку с запятой, дать возврат каретки, и ждать дальнейших сообщений. Если же Вы чувствуете себя еще неуверенно, прервите программу клавишей Ctrl-C и подайте команду снова;

```
File Access Error In prog1 . PAS; Code: 1032; Status: 2
```

```
Compiler Cannot Continue
```

(не могу прочесть файл `prog1 . pas`)

Вы все-таки не так назвали файл с программой, как мы договорились, или не так набрали его имя в команде компиляции. Во всяком случае, файла с таким именем и расширением не найдено. Разберитесь, в чем дело: может быть Вы забыли записать на диск набранный в редакторе текст и т. п.;

```
11      3 write ( 'Программа 1 выполнена!'
```

```
3 — — — — — Warning 169 Insert ) ^
```

```
Pass One 1 Warnings Detected
```

(В строке 3 обнаружена грамматическая ошибка.)

Компилятор нашел грамматические ошибки в программе. Советуем распечатать текст программы, набрав `copy prog1 . pas rpn`, и тщательно сверить его с тем, что в книжке. Единственная полезная информация, которую можно извлечь из сообщения об ошибке (кроме самого факта сообщения), — это номер строки, в которой обнаружена самая первая ошибка. Этот номер, как видите, повторен дважды, перед ошибочной строкой и перед сообщением об ошибке. Сообщение об ошибке следует понимать так: часть текста, который приказано откомпилировать, от начала до места, помеченного значком «^» включительно, не может быть началом правильной программы на Паскале. Это не означает, однако, что ошибка была сделана именно

в этом месте. Вы могли что-то не так набрать и ранее, прост компилятор понял, что Вы неправы, лишь дочитав программ до этого места. Не исключено, что сообщений будет несколько даже довольно много — игнорируйте все, кроме первого, так как все они могут быть порождены единственной ошибкой. Исправьте ошибку с помощью текстового редактора, и снова прикажите компилировать.

Но вот наконец-то программа PAS1 отработала без единого замечания, и на дисплее снова появилось приглашение ОС. Что же произошло? Наберите DIR, и Вы увидите, что в текущем каталоге появилась пара новых файлов: PASIBF . BIN и PASIBF . SYM. Эти файлы содержат перевод вашей программы на промежуточный язык, так сказать «полуфабрикат» для изготовления программы на машинном языке. Изготовлением этой программы занимается вторая из программ компилятора — программа PAS2 . EXE. Вызывается она совсем просто:

PAS2

(изготовить из имеющегося полуфабриката файл в машинном формате)

Процесс выполнения этой команды выглядит точно так же, как и команды PAS1. На экране на этот раз не появляется никакого приглашения, а просто через некоторое время сообщается следующее:

Code Area Size= #002B (43)

Cons Area Size= #0019 (25)

Data Area Size= #0010 (16)

Pass Two No Errors Detected.

Программа готова. Если Вы посмотрите на содержимое Вашего каталога, то обнаружите, что файлы PASIBF исчезли, зато появился файл PROG1 . OBJ. Конечно, от этой программы тоже могут поступить сообщения о каких-либо ошибках, но в случае наших простых программ это маловероятно. Раз уж Вы забудете выполнить первую фазу компиляции (т. е. — PAS1) перед выполнением второй. Тогда ждите примерно такого сообщения:

File access error in file PASIBF . SYM, error code 1032

Итак, Ваша программа откомпилирована и результат компиляции помещен в файл PROG1 . OBJ. Но и это еще не все. Теперь Вам предстоит преобразовать этот файл в выполняемую программу, добавив к программе, написанной Вами, описание всех использованных в ней процедур. Вы можете возразить, сказав, что никаких процедур Вы тут не вызывали — программа-то всего в одну строчку. На самом деле любая программа обращается к ряду стандартных процедур, о которых пользователь не подозревает. Эти обращения добавляет компилятор. Например, каждой программе придаются стандартные процеду-

ры обращения к ОС во время работы: для работы с файлами, для работы с экраном и клавиатурой, для правильного завершения программы и передаче управления ПК обратно в ОС. Все эти стандартные блоки находятся в файлах с расширением LIB, которые мы копировали при установке компилятора. Они закодированы в машинном формате, так что компиляция им не нужна, а вот добавить их к Вашей программе необходимо. Этим занимается специальная программа, работу с которой мы сейчас и начнем.

Редактор связей. Получением окончательного варианта программы из объектного модуля (так называется результат работы компилятора, который он помещает в файл с расширением .OBJ) занимается программа «редактор связей», часто называемая на программистском жаргоне «линкер» (от английского to link — связывать). Вот как запускается эта программа:

```
link prog1;
```

Смысл этого такой:

link — создай выполняемую программу...

prog1 — ...из объектного модуля, находящегося в файле prog1 . obj (заметьте, что у редактора связей предполагаемое расширение файла иное, чем у компилятора)...

«;» — ...и не задавай никаких вопросов.

Если файл link . exe действительно есть на диске, на дисплее покажется «товарный знак» редактора связей, примерно такой:

```
Microsoft (R). Overlay Linker Version 3 . 60
```

```
Copyright (C) Microsoft Corp 1983, 1984, 1985, 1986.
```

```
All rights reserved.
```

Подобно компилятору, редактор начнет задавать вопросы, если Вы забудете поставить точку с запятой. Опять-таки, если после выдачи «товарного знака» через некоторое время высветилось приглашение ОС — это значит, что все в порядке. Возможны и сообщения об ошибках. Вот некоторые из них:

```
Cannot find file PROG1 . OBJ
```

```
Change diskette and press <ENTER>
```

```
(не нашел файла PROG1 . OBJ)
```

Видимо, компилятор не создал файла с объектным модулем. А может быть, вы сообщали компилятору другое имя?.. Во всяком случае, пока Вы (с помощью компилятора) не создадите объектный файл, выполнять команду link бессмысленно.

```
Cannot find library: имя . LIB
```

```
Enter new file spec:
```

```
(не нашел библиотеки с заданным именем)
```

Редактор связей начал поиск описаний процедур, использованных в вашей программе, но не обнаружил тех файлов с расширением LIB, в которых они должны храниться. Проверьте, на месте ли эти файлы. Кроме того, подайте команду SET, и убедитесь, что среди напечатанных строк системного окружения есть такая:

LIB=C:\LIB

Строка «LIB=путь» предназначена специально для редактора связей: из нее он узнает, в каком из каталогов следует искать библиотечные файлы. Заметьте, что вид строки «PATH=» никак не влияет на его поиски, так что бессмысленно указывать путь к библиотечным файлам в строке «PATH».

Unresolved externals:

— имя in file(s):

PI . OBJ

(в тексте программы содержится вызов процедуры, которой нет в библиотеке стандартных процедур)

Во второй строчке этого сообщения указывается имя процедуры, вызов которой есть в Вашей программе, но ее описание в стандартных библиотечных файлах отсутствует. Вы, видимо, ошиблись, набирая имя процедуры в исходном тексте программы. Компилятор *не проверяет*, является ли вызываемая процедура стандартной — он вообще не знает, какие процедуры стандартные, а какие нет. Поэтому при компиляции никаких сообщений не было. Придется исправить ошибку в программе и начать все сначала, т. е. вновь откомпилировать ее и затем выполнить команду link.

Если, наконец, редактор связей проработал без единого сообщения об ошибке, в текущем каталоге должен появиться файл PROG1 . EXE. Он, как и положено файлу с расширением EXE, содержит выполняемую программу. Достаточно лишь набрать PROG1 и дать возврат каретки — и на экране появится текст: «Программа 1 выполнена!».

Надеемся, что здравый смысл и приведенные выше напутствия помогли Вам справиться хотя бы с каким-нибудь компилятором из описанных (или с каким-то еще — неважно). Поэтому мы не будем ничего больше рассказывать про компиляцию, а приведем примеры программ на языке Паскаль, попутно изучая и сам язык. Заметим, что дальнейший текст ни в коем случае не следует рассматривать как «полный толковый словарь» по языку Паскаль. Язык Паскаль гораздо богаче, мы же приводим лишь краткий «словарь-разговорник», содержащий самое необходимое для того, чтобы начать работать на Паскале.

Типы переменных. Итак, продолжаем перевод на Паскаль программ в гл. 4. Вот как будет выглядеть программа 2, обменивающая значения двух чисел, в «паскалевском» исполнении:

```
program Prog2 (input, output);
var a, b, x : integer;
begin
  a := 2; b := 3;
  writeln ('До обмена A=', a, 'B=', b);
  x := a; a := b; b := x;
  write ('После обмена A=', a, 'B=', b);
end.
```

Видно, что перед нами опять практически дословный перевод написанной ранее программы (вспомните, как она выглядела). Посмотрим, что изменилось в результате такого перевода.

Описание типов переменных располагается между заголовком программы и словом `begin`, и ему предшествует ключевое слово `var` (это от английского *variings*, что означает «переменные»). Названия типов также переведены на английский. Приведем перевод всех употребляемых нами названий типов на Паскаль:

цел — `integer`,
вещ — `real`,
симв — `char`,
лог — `boolean`.

В одной программе могут быть, естественно, описаны переменные разных типов. Все эти описания помещаются между ключевыми словами `var` и `begin`. Например, в начале какой-нибудь программы может встретиться такой набор описаний:

```
var
  a, b, c : real;
  n, q1, следующий : integer;
  новсимв : char;
  найден __ x : boolean;
begin
```

и дальше текст программы

Русские буквы мы использовали здесь для наглядности. В действительности ни один из описанных нами компиляторов не допускает русских букв нигде, кроме выводимых на печать строк и комментариев. А вот символ подчеркивания уравнен в правах с латинской буквой и может встречаться в Ваших программах.

Мы использовали еще один вид команды печати `writeln`. От команды `write` она отличается лишь тем, что после ее выполнения происходит переход на другую строку. Это позволило нам опустить команду перевода строки перед второй печатью. Заметьте, что команда `writeln` без параметров эквивалентна нашей команде «перевести строку».

Команды ввода. Новая команда, которая появляется в программе 3, — команда получения символа с клавиатуры. С этой командой в Паскале дело обстоит плохо — ее там просто нет. Точнее, ее нет в стандартном Паскале. Разработчики компиляторов обычно вводят в Паскаль дополнительные команды, в том числе и команды, позволяющие прочесть символ с клавиатуры. Поскольку невозможно описать все варианты этих расширений во всех компиляторах, мы не будем описывать их вовсе, порекомендовав читателю обратиться к руководству по соответствующему компилятору. Но раз уж мы пообещали перевести все пронумерованные в гл. 4 программы, слово придется сдержать. Приведем переводы программы 3 на языки описанных нами компиляторов. Вот они:

```
{программа для компилятора Турбо}
program PROG3 (input, output);
var
  C : char;
begin
  while not KeyPressed; read (kbd, C);
  write (c, c);
end.
{программа для компилятора Microsoft}
program PROG3 (input, output);
var
  c : char;
  function dosxqq
    (ah, dx : integer) : char ; extern;
begin
  c := dosxqq (7,0);
  write (c, c);
end.
```

Строки в фигурных скобках — это комментарии. Как видите, они записываются точно так же, как писали их мы. Мы никак не будем комментировать эти две программы, порекомендовав лишь дословно воспроизвести подобные записи в Ваших программах, если Вам понадобится команда «получить символ». Изменяться может лишь имя переменной, в которую помещается введенный символ. Единственное, о чем стоит, пожалуй, упомянуть: второй вариант перевода не нарушает правил стандартного Паскаля. В нем просто используется нестандартная функция, описание которой находится не в самом файле, а где-то еще. Такая функция называется *внешней* по отношению к данной программе. Правила языка допускают использование внешних функций, нужно только указать типы всех параметров этой функции и тип возвращаемого результата. Описания всех функций — и внешних в том числе — принято помещать сразу после описания всех переменных (но перед

словом begin!). Мы еще приведем примеры описания внешних процедур и функций в программах на Паскале.

Команда, эквивалентная нашей команде «ввести», к счастью, в Паскале есть. Она называется readln и употребляется точно так же, как и «ввести». Например, перевод на Паскаль программы 4 будет выглядеть так:

```
program PROG4 (input, output);
{печать квадрата введенного числа}
var x : integer;
begin
    writeln;
    write ('Введите целое число');
    readln (x);
    writeln; { =перевести строку}
    write ('Квадрат числа', x, ' равен', x * x);
end.
```

Параметрами команды «ввести», как Вы помните, могут быть произвольные арифметические выражения. Довольно простое выражение, означающее квадрат значения переменной «х», мы видим и среди параметров write в данной программе. Расскажем о правилах записи арифметических выражений на Паскале чуть подробнее.

Как и в нашем учебном языке, в Паскале можно составлять выражения из констант и значений переменных. Естественно, набор операций при этом строго фиксирован.

С целыми числами можно производить такие операции: «+» (сложение), «-» (вычитание), «*» (умножение), div (деление нацело — дробная часть, если она есть, отбрасывается), mod (остаток от деления; запись A mod B означает остаток от деления A на B. Будьте, однако, осторожны: лишь при делении положительных чисел значение остатка соответствует математическому определению!).

Вещественные числа можно складывать, вычитать, умножать (обозначения те же), делить (обозначение «/»). Как целые, так и вещественные числа можно сравнивать — результат сравнения будет иметь логический тип. Операции сравнения пишутся так:

= (равно),	<> (не равно),
> (больше),	< (меньше),
>= (больше или равно),	<= (меньше или равно).

Символы тоже можно сравнивать — но только на «равно» и «не равно» (обозначения те же). Операции сравнения используются там, где необходимы проверки, т. е. в конструкциях выбора и цикла (чуть позже мы скажем, как выглядят эти конструкции в Паскале).

Наконец, над логическими значениями можно производить операции `not` (не), `and` (и), `or` (или). Таким образом, мы можем составлять выражения вроде таких:

```
(a + b) / c
(следующий * n + q1) div (n + q1)
(найден — x or not (a = b) ) and (n < q1)
```

Если считать, что использованные только что переменные описаны, как в приведенном несколько раньше примере, то первое выражение даст в результате значение действительного типа, значение второго выражения — целое число (напомним, что `div` — это деление нацело), третье выражение — логического типа.

Порядок выполнения операций в выражении определяется скобками. Кроме того, действуют обычные правила вроде того, что умножение и деление выполняются раньше сложения и вычитания, и т. п. Вряд ли нужно разучивать, какая операция выполняется раньше, а какая позже: ведь если есть сомнения, всегда можно поставить лишнюю пару скобок. Поверьте, что программа от этого не станет менее эффективной. Выражения в Паскале появляются там же, где они появлялись в нашем языке — в правой части команды присваивания и в операторе вывода. А логические выражения, кроме того, появляются в качестве условий в конструкциях выбора и цикла. Кстати, пришла пора рассказать, как выглядят эти конструкции на Паскале.

Программные конструкции. В следующей программе — программе 5 — встречается программная конструкция выбора. Посмотрите, как она выглядит на Паскале:

```
program PROG5 (input, output);
var
  a, b, max : integer;
begin
  writeln;
  write ('Введите два целых числа:');
  readln (a, b);
  if a > b then begin
    max := a
  end else begin
    max := b
  end;
  writeln;
  write ('Максимальное из введенных чисел —', max)
end.
```

Заметим, что мы предприняли одно небольшое усовершенствование, введя оба числа при помощи одной команды `readln`. Такая запись в Паскале тоже возможна. Но главное новшество в этой программе — это, конечно, конструкция выбора. Проиллюстрируем общую схему перевода этой конструкции

на Паскаль. Слева дана конструкция выбора на нашем учебном языке, справа — ее перевод:

ЕСЛИ условие ТО	if условие then begin
действие A1	действие A1;
действие A2	действие A2;
действие A3	действие A3;
ИНАЧЕ	end else begin
действие B1	действие B1;
действие B2	действие B2;
действие B3	действие B3;
КОНЕЦ ЕСЛИ	end

Предполагается, что все использованные в конструкции действия и условия также переведены на Паскаль.

Следующая из построенных нами программ иллюстрирует применения конструкции цикла. Вот как выглядит программа 6, если ее записать на Паскале:

```

program PROG6 (input, output);
var
  x, max : integer;
  {последнее и максимальное из введенных чисел,
   соответственно}
  answer : char; {ответ : символ}
  {принимает значение «n», если ввод чисел не закончен}
begin
  writeln;
  write ('Введите число:');
  readln (x);
  writeln;
  write ('Это все? (y/n) ');
  readln (answer);
  max := x;
  {max = максимальное из всех введенных чисел}
  {ответ = «n», если есть еще числа}
  while answer = 'n' do begin
    {max = максимальное из всех введенных чисел}
    {есть еще числа}
    writeln;
    write ('Введите число:');
    readln (x);
    writeln;
    write ('Это все? (y/n) ');
    readln (answer);
    {max = максимальное из всех введенных чисел,
     не считая последнего}
    {x = последнее введенное число}
    if max < x then begin
      max := x
    end
  end
end

```

```

end
{max = максимальное из всех введенных чисел}
{ответ = «п», если есть еще числа}
end;
{max = максимальное из всех введенных чисел}
{чисел больше нет, т. е. max = максим. из всех}
writeln;
write ('Максимальное из введенных чисел —', max)
end.

```

Как видим, конструкция цикла переводится на Паскаль таким образом:

ПОКА	условие	ПОВТОРЯТЬ	while	условие	do begin
	действие 1			действие 1;	
	действие 2			действие 2;	
	действие 3			действие 3;	
КОНЕЦ ПОКА			end		

Процедуры. Приведенные в гл. 4 процедуры переводятся на Паскаль почти дословно. Запишем, например, полный текст программы вычисления максимума с использованием процедур. Вот что у нас получится:

```

program PROG6 (input, output);
var
  x, max : integer;
  answer : char;
  procedure input__number
    (var number : int, var letter: char);
  begin
    writeln;
    write ('Введите число:');
    readln (number);
    write ('Это все? (y/n) ');
    readln (letter)
  end
begin
  input__number (max, answer);
  while answer = 'n' do begin
    input__number (max, answer);
    if max < x then begin
      max := x
    end
  end;
  writeln;
  write ('Максимальное из введенных чисел —', max)
end.

```

Перечислим отличия принятого в Паскале способа описания от того, которым пользовались мы:

текст описания помещается внутри текста самой программы, после описаний всех переменных, но перед словом **begin**;

если среди параметров процедуры встречаются выходные, им должно предшествовать слово `var`, являющееся, как Вы помните, сокращением английского слова «переменная». Поскольку на месте выходного параметра при обращении к этой процедуре должна стоять переменная, слово это вполне уместно;

тело процедуры устроено так же, как и тело основной программы. Там могут быть описания каких-то промежуточных переменных (тогда нужно написать слово `var` сразу после заголовка; заметьте, что это слово употребляется в двух различных местах).

Если при описании процедуры в списке параметров нет слова `var` перед каким-либо параметром, этот параметр считается *входным*. При обращении к процедуре из основной программы на месте входного параметра может быть написано любое выражение того же типа, что и входной параметр. В качестве примера приведем описание процедуры «напечатать __ несколько __ раз» (`print several times` — в переводе на английский), печатающей заданное действительное число заданное количество раз:

```
procedure print __ several __ times (R : real, N : integer);  
  var K : integer;  
begin  
  K := 0;  
  while K < N do begin  
    writeln (R); K := K + 1  
  end  
end.
```

Паскаль позволяет определять не только новые команды, но и новые функции, и использовать их потом при записи выражений. Например, функция, вычисляющая гипотенузу треугольника по его двум катетам, может быть записана так:

```
function gip (x, y : real) : real;  
begin  
  gip := sqrt (x * x + y * y)  
end
```

Перечислим основные отличия процедур-функций от обычных процедур:

вместо ключевого слова `procedure` пишут ключевое слово `function`;

имя функции имеет тип. Он указывается после списка параметров;

в тексте описания функции переменная с именем этой функции должна получить какое-нибудь значение. Это значение и является единственным результатом обращения к функции;

процедура-функция *не изменяет никакие переменные программы*. Поэтому все ее параметры — входные (перед ними нельзя писать `var`).

На этом мы закончим введение в язык Паскаль, еще раз повторив, что это лишь введение. Следующий сравнительный обзор двух рассмотренных нами компиляторов предназначен читателям, уже имеющим (или уже приобретшим) опыт работы на Паскале.

К недостаткам языка Паскаль, снижающим его конкурентоспособность по сравнению с другими языками программирования, явилось отсутствие стандарта на интерфейс с ОС. Классический пример: в стандартном Паскале имеется тип `file` (файл) и команды работы с ним. Так, Вы можете поместить в вашей программе такое описание:

```
stream : file of integer;
```

объявив тем самым, что переменная `stream` является файлом для хранения целых чисел. При этом допустим обычный набор команд последовательного доступа, т. е. команды записи в файл, чтения из файла, перемотки файла и т. д. Однако в стандарте никак не оговорена связь файловой переменной с реальными файлами. Естественно, разработчики компиляторов решают эту проблему, доопределяя соответствующие команды, но всякий делает это по-разному. Такая относительная неполнота Паскаля привела к тому, что всякий компилятор Паскаля имеет, по сути дела, свой особый входной язык, так что о переносимости программ одного компилятора на другой, а тем более на другие ЭВМ, говорить не приходится. Есть, впрочем, основания считать, что Турбо-Паскаль станет фактическим стандартом для ПК типа IBM PC.

В компиляторе Microsoft (и в версии 4 Турбо-Паскаля, которая появилась совсем недавно) есть возможность использования внешних процедур, которые могут быть написаны на любых языках, соблюдающих определенные соглашения о передаче параметров. Ряд процедур по связи с ОС имеется в поставляемых вместе с компилятором библиотеках, хотя этот набор не так велик, как, скажем, набор стандартных процедур для языка Си. Однако допускается использование библиотечных процедур, написанных на Си. Приведем простой пример — использование команды `system` стандартной библиотеки Си. Следующая программа распечатывает на экране имена всех файлов текущего каталога, имеющих расширение `PAS`:

```
program dirpas (input, output);
var
  i : integer;
  function system : integer [c, varying]; extern;
begin
  i := system ( ads ( 'dir * . pas' * chr (0) ));
end.
```

Функция `system` позволяет выполнить переданную ей командную строку, как если бы она была введена в режиме диалога с ОС. Строка должна заканчиваться символом с ко-

дом 0. Как видим, после заголовка процедуры следует описание этой функции в качестве внешней, со специальным указанием, что вызов ее должен происходить в стандарте языка Си. Поскольку в стандартной библиотеке компилятора Паскаля эта функция не содержится, должен быть явно указан файл, содержащий ее описание. В данном случае это файл LLIBC.LIB, поставляемый вместе с компилятором Microsoft Си, и содержащий описание большинства доступных из Си команд. Инструкция по компиляции и сборке такой Паскаль-программы будет выглядеть так:

```
PAS1 DIRPAS;
PAS2
LINK DIRPAS , , LLIBC
```

Так же легко связывается Паскаль и с фортрановскими подпрограммами. Более того, в последнем случае не требуется даже специального указания, так как Паскаль и Фортран используют одинаковые соглашения о передаче параметров.

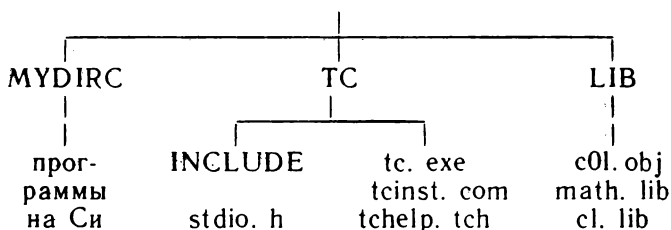
Поскольку в описываемых версиях Паскаля не содержится средств для работы с устройствами ПК, для использования ряда возможностей, описанных нами в гл. 5, придется обращаться к подпрограммам библиотеки языка Си либо внимательно изучить руководство к имеющемуся у Вас Паскаль-компилятору — может быть его входной язык расширен и включает как раз нужные Вам средства.

6.2. ЯЗЫК СИ

К настоящему моменту разработано довольно много различных компиляторов с языка Си, в том числе и для IBM-совместимых машин. Мы приведем пример работы с двумя из довольно распространенных компиляторов. Си.

Редактор-компилятор Турбо-Си фирмы Borland. Поставляется на четырех дискетах. Для его работы требуется не менее 200 Кбайт свободной оперативной памяти. Рекомендуется создать на диске такую структуру (большими буквами выделены названия каталогов):

Корневой каталог



Кроме того, следует указать путь к этому каталогу, добавив к команде PATH в файле AUTOEXEC строку «C:\TC». Собственно редактор-компилятор помещается в файле tc. exe, однако прежде чем его использовать, следует предварительно «подготовить» этот файл к работе на жестком диске в выбранном каталоге. Опишем один из вариантов процедуры установки параметров. При этом наше описание будет сугубо прагматическим: «если сделать так, то будет достаточно хорошо». Однако мы уверены, что не пройдет и получаса, как Вы сами поймете, как управлять этим компилятором. Этому способствуют удобная система меню. Кроме того, Вы в любой момент можете получить краткую справочную информацию, нажав F1. Итак, вот последовательность действий по инициализации редактора-компилятора.

1. Передайте в каталог TC.

2. Наберите TCINST и дайте возврат каретки. На экране появится примерно такое меню:

```
Turbo C directories
Editor commands
Setup environment
Display mode
Colors
Resize windows
Quit/save
```

Нажатие каждой из заглавных букв означает переход к соответствующему подменю. Нажав T, мы окажемся в меню задания рабочих каталогов компилятора, которое выглядит так:

```
Include directories
Library directories
Output directory
Turbo C directory
Pick file name
```

Для созданной нами файловой структуры необходимо задать такие параметры:

```
Include directories — c:\tc\include,
Library directory — c:\lib,
Turbo C directory — c:\tc.
```

Для задания этих параметров достаточно нажать первую букву соответствующего варианта (L, I, T), после чего ввести указанную строку, завершив ввод возвратом каретки. Нажав ESC, вернемся в основное меню, после чего нажмем Q. В результате файл tc. exe будет изменен в соответствии с заданными параметрами.

3. Теперь придется подготовить некоторые параметры в компиляторе. Для этого войдем в него, набрав TC, а затем нажмем Alt-O, войдя в меню Options (параметры). Для

того чтобы успешно выполнить приведенные в этом параграфе программы, следует изменить один из параметров компилятора, так что войдем в подменю Compiler (догадайтесь сами, как это сделать) и изменим там параметр Model на Large. Этот параметр определяет, как Си будет интерпретировать адреса памяти. Способ, задаваемый этой моделью, совпадает с описанным нами в § 5.2. Он не всегда приводит к самой эффективной выполняемой программе, но зато является самым простым.

4. Выполним вариант Store options в меню Options, запоминая установленные параметры в специальном файле. Теперь можно нажать Alt-X, выйдя в ОС, и перейти в свой каталог. Всю дальнейшую работу можно производить там.

Компиляция программы под управлением этого компилятора выглядит крайне просто. Следует лишь набрать на клавиатуре ТС и имя Вашей программы, например PROG1, Вы окажетесь в компиляторе. Если файл с программой уже был создан, достаточно нажать Alt-R — и через считанные секунды Вы увидите на экране результат работы Вашей программы либо сообщения об ошибках, которые появятся в нижней части экрана. При этом система автоматически вернется в текстовый редактор прямо к тем строкам программы, в которых была обнаружена ошибка, а эти строки будут выделены цветом.

Работа с компилятором возможна и в случае, если на Вашем ПК отсутствует винчестерский диск, но тогда набор параметров, задаваемых программой TCINST, должен выглядеть несколько иначе:

Include directories — b:

Library directory — b:

Turbo C directory — a:

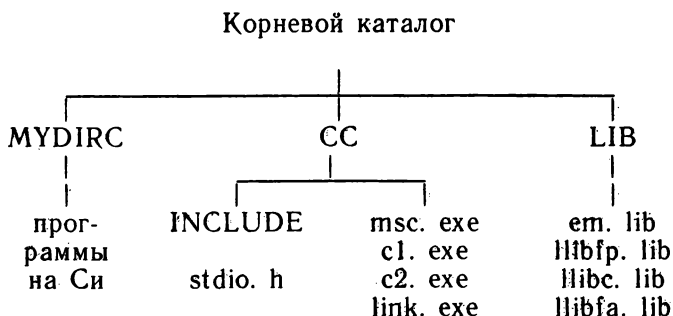
Возможен следующий порядок работы при отсутствии жесткого диска:

1. Подготовьте дискету для своих программ. Рекомендуется записать на нее и ОС. Вставьте подготовленную дискету в дисковод A: и загрузите ОС.

2. В дисковод B: вставьте дискету, содержащую файл ТС. EXE. Запустите компилятор, набрав B : ТС. После того как редактор-компилятор загрузился в память ПК, содержащая его дискета больше не нужна для работы. Вместо нее вставьте в дисковод B: дискету с файлами, необходимыми во время компиляции. В комплекте поставки обязательно есть дискета, содержащая все такие файлы (это файлы с расширениями H и LIB). Теперь Вы можете откомпилировать и выполнить любую программу со своего диска. В процессе компиляции возможны обращения к диску B:, но только для чтения.

Компилятор Microsoft Си. Получившего этот компилятор поражает прежде всего обилие дискет с различными программами. Но на самом деле большинство из них нам на первых порах не понадобятся (а честно говоря, вряд ли вообще когда-ли-

бо потребуются). Однако даже минимальный объем этой системы настолько велик, что пользоваться ею на машине без жесткого диска вряд ли целесообразно. Если же такой диск на Вашей машине имеется, следует скопировать на него необходимые файлы с дискет. Для первых упражнений с компилятором Си мы рекомендуем создать на диске такую структуру (большими буквами выделены названия каталогов):



Упражнение. Что надо для этого сделать?

Решение. Вот одна из возможных последовательностей команд:

```

c:
cd\
md cc
md lib
cd cc

```

Теперь следует вставить дискету, содержащую файлы с расширением .exe и набрать:

```

copy a : msc. exe
copy a : c?. exe
copy a : link. exe
md include
cd include

```

Теперь надо вставить дискету с .H-файлами (т. е. файлами с расширением .H) и набрать:

```

copy a : *.h
cd\lib

```

Наконец, вставить дискету с нужными .LIB-файлами и набрать:

```

copy a : *. lib
cd\

```

Как видим, приходится набирать на клавиатуре довольно много команд. Это как раз тот случай, когда удобнее воспользоваться файловым монитором.

Итак, в созданный каталог CC требуется скопировать указанные файлы (в них, в частности, содержится программа-

компилятор), в подкаталог LIB — все или по крайней мере перечисленные файлы с расширением LIB. Если позволяет память, то виртуальный диск на 64 Кбайта (больше не надо) также не помешает. Кроме того, в файл AUTOEXEC. BAT придется добавлять такие команды:

```
set lib=c:\lib
set include=c:\cc\include
path=то, что уже есть; c:\cc
set tmp=d:\
```

Последняя команда, как Вы помните, полезна при наличии виртуального диска и недопустима при его отсутствии. Не забудьте перезагрузить ОС после изменения файла AUTOEXEC — иначе это изменение никак не проявится.

Достроив файловую систему таким образом, перейдем в свой каталог, т. е. в тот, в котором мы намерены хранить свои программы на Си. Стоит, быть может, завести отдельный каталог для этой цели. Пусть он называется, например, MYDIRC. Всю дальнейшую работу мы будем выполнять, находясь в этом каталоге.

Переведем на Си первую из написанных в гл. 4 программ. Вот как выглядит этот перевод (для сравнения вначале дан оригинал):

ПРОГРАММА 1

напечатать («Программа 1 выполнена!»)

КОНЕЦ ПРОГРАММЫ

```
main () {
    printf («Программа 1 выполнена!»);
}
```

Не надо быть профессиональным лингвистом, чтобы установить соответствие между обоими текстами. Отметим лишь три особенности:

Строка 1: программа на Си не имеет ни имени, ни номера: строка «main()» — это лишь перевод слова ПРОГРАММА. Именно такой строкой будут начинаться все наши программы на Си.

Строка 2: после каждой команды в Си ставится точка с запятой.

Строка 3: слова КОНЕЦ ПРОГРАММЫ переводятся на Си единственным символом — закрывающей фигурной скобкой. Удобно представлять, что она закрывает фигурную скобку, открытую в первой строке.

Как выполнить эту программу? Для начала, разумеется, надо ввести ее в компьютер. Это можно сделать с помощью любого текстового редактора. Сделайте это, и пусть текстовый файл, который у Вас получится, называется PROG1. C. Хранить программы на Си в файлах с расширением .C — традиция, которой будем следовать и мы. Приятно, что и компилятор об этой традиции осведомлен — вскоре Вы увидите, в чем это

выражается. Ну а имя PROG1 мы выбрали просто так — надо ведь дать файлу какое-то имя. Вернувшись из редактора в ОС, подадим такую команду:

```
msc/AL prog1;
```

Понимать это надо так:

msc/AL — откомпилируй, т. е. переведи с языка Си на машинный язык...

prog1 — ...программу, находящуюся в файле prog1. с (поскольку мы не указали расширение, компилятор добавил традиционное расширение .C)...

«;» — ...и не задавай лишних вопросов.

Если все вышеописанное было сделано безошибочно, компилятор выведет на дисплей свой «товарный знак»:

```
Microsoft (R) C Compiler Version 4. 00
```

```
Copyright (C) Microsoft Corp 1984, 1985, 1986.
```

```
All rights reserved.
```

Затем ПК «подумает» немного (не более минуты) и высветит приглашение ОС. Запомните — признаком удачной работы компилятора является то, что на дисплее не появляется никаких сообщений. А вот если что-то было сделано не так — тогда непременно ждите комментариев на экране. Вот несколько возможных сообщений после подачи команды компиляции:

```
Bad command or file name
```

(здесь: не нашел такого файла)

Это сообщение Вам, видимо, знакомо. Возможно, Вы неправильно набрали команду, например вместо обычной косой черты ввели обратную косую черту — проверьте! Если команда набрана без ошибок, а это сообщение все равно появляется, это означает, что Вы либо забыли скопировать на диск (или скопировали не туда) файл msc. exe, либо указали путь к каталогу CC командой PATH в файле AUTOEXEC. Набрав команды PATH (без параметров и знака равенства), а также DIR\CC Вы быстро разберетесь, в чем дело. Если эта трудность преодолена, и на экран выводится «товарный знак» компилятора — считайте, что победы над файловой системой Вы добились — программа-компилятор заработала. Однако и от нее могут поступить неприятные сообщения. Вот несколько примеров сообщений, которые могут появиться уже после начала работы компилятора:

```
Command line warning 10: ignoring unknown flag 'al'
```

(непонятно, что за флаг)

Вы наверно набрали флаг /AL, используя маленькие буквы вместо заглавных. Что бы ни делал после этого ПК, прервите его, нажав Ctrl-C, и дождавшись приглашения ОС, введите команду правильно.

```
Object filename [prog1. OBJ]:
```

(куда поместить результат компиляции [может быть, в файл prog1. obj]?:)

Это уже вопрос, а не утверждение. Видимо, набирая команду, мы забыли про точку с запятой — приказ компилятору не спрашивать ни о чем (своих вопросов хватает!). Результат, как видим, не замедлил сказаться. Но еще не поздно — нажмите теперь точку с запятой, а затем дайте возврат каретки — и ждите дальнейших сообщений. Если же Вы настаиваете на чистоте эксперимента, прервите программу и введите команду снова;

Command line warning 20: could not execute 'cl. exe'.

Please insert diskette and hit any key

(не могу найти файл C1.EXE)

Компилятор Си состоит из нескольких частей, которые он сам подгружает с диска по мере надобности. Сейчас понадобилось подгрузить файл C1.EXE, но такой файл не обнаружен. Прервите программу-компилятор и проверьте, есть ли в каталоге CС этот файл;

Command line fatal error 19: could not execute 'cl. exe' (не могу выполнить файл C1.EXE)

Файл C1.EXE найден, но выполнить находящуюся в нем программу не удастся. Скорее всего, программа не умещается в оперативной памяти (убедитесь, набрав DIR, что файл этот и впрямь огромных размеров). Придется как-то так изменить ОС, чтобы свободной оперативной памяти стало больше. Уберите из памяти ненужные резидентные программы, которые допускают такую возможность. Если это не помогло, отредактируйте CONFIG. SYS, уменьшив до минимума размер виртуального диска, и AUTOEXEC. BAT, убрав команды загрузки ненужных резидентных программ, и затем перезагрузите систему;

fatal error 13: cannot open 'prog1. C'

(не могу прочесть файл prog1. c)

Вы не так называли файл с программой, как мы договорились, или не так набрали его имя в команде компиляции. Во всяком случае, файла с таким именем и расширением не найдено. Разберитесь, в чем дело: может быть вы забыли записать на диск набранный в редакторе текст и т. п.;

prog1. C(3) : error 143: syntax error : missing ';' before '}'

(в файле prog1. c, в строке 3, обнаружена грамматическая ошибка)

Компилятор нашел грамматические ошибки в программе. Советуем распечатать текст программы, набрав PRINT PROG1. C, и тщательно сверить его с тем, что в книжке. Единственная полезная информация, которую можно извлечь из сообщения об ошибке (кроме самого факта сообщения) — это номер строки, в которой обнаружена самая первая ошибка. Вон он, этот номер, в скобках сразу после имени файла с нашей ошибочной программой. Не исключено, что сообщений будет несколько, даже довольно много — игнорируйте все, кроме перво-

го, ибо, скорее всего, все они порождены единственной ошибкой. (Это одна из неприятных особенностей, характерных для данной модели компилятора.) Исправьте ошибку с помощью текстового редактора и снова запустите компиляцию.

Но вот, наконец-то, компилятор отработал, «не сказав ни слова», и на дисплее снова появилось приглашение ОС. Что же произошло? Наберите DIR, и Вы увидите, что появился новый файл PROG1.OBJ. Это и есть результат работы компилятора и Ваших усилий. Правда, результат не окончательный — предстоит еще преобразовать этот файл в выполняемую программу, добавив к программе, написанной Вами, описания всех использованных в ней процедур. Эти описания как раз и находятся в директории LIB, в файлах с расширением LIB. Язык Си интересен тем, что элементарных команд там всего две — команда присваивания и команда обращения к процедуре. Зато набор стандартных процедур довольно велик — посмотрите, какие размеры имеют LIB-файлы. Итак, вторая строка нашей программы — вызов стандартной процедуры printf, эта процедура должна быть найдена в одной из библиотек и добавлена к нашей программе.

Для этого придется воспользоваться специальной программой «редактор связей». Работа с этой программой подробно описана в предыдущем параграфе, в подразделе, который так и называется: редактор связей. Если Вам незнакома эта программа, мы рекомендуем Вам прерваться и прочитать этот раздел, и лишь затем возвращаться к дальнейшему чтению.

Итак, мы надеемся, что вам удалось получить выполняемую программу, сообщающую о факте своего выполнения на дисплее всякий раз, когда Вы ее вызываете. Теперь продолжим перевод на Си программ, написанных в гл. 4, заодно изучая и сам язык.

Типы переменных и команда печати. Вот как будет выглядеть наша вторая программа, обменивающая значения двух чисел, в переводе на язык Си:

```
main () {
    int a, b, x;
    a=2; b=3;
    printf («До обмена A=%d B=%d», a, b);
    x=a; a=b; b=x;
    printf («\nПосле обмена A=%d B=%d», a, b);
}
```

Видно, что перед нами опять практически дословный перевод написанной ранее программы (вспомните, как она выглядела). Посмотрим, что изменилось в результате такого перевода:

1. Иначе выглядит описание типов переменных: «цел» переводится на Си как int — сокращение от английского слова integer, как раз и означающее «целый», причем в Си название типа пишется не после списка переменных, а перед ним, а спи-

сок отделяется от названия типа по крайней мере одним пробелом. Завершается список точкой с запятой. Перечислим несколько стандартных типов, допустимых в Си:

`int` — переменные этого типа могут принимать целые значения в диапазоне от -32768 до 32767 ;

`unsigned int` — название типа состоит из двух слов и переводится как «целое без знака». Переменные этого типа могут принимать значения от 0 до 65535 ;

`unsigned char` — соответствует типу «симв» в языке, которым пользовались мы. Переменные этого типа предназначены для хранения символов. На деле это означает, как вы помните, что в переменной содержится некоторое целое число, являющееся номером соответствующего символа. Так вот, диапазон значений таких переменных простирается от 0 до 255 ;

`long int` — переводится на русский язык как «длинное целое». Переменная такого типа занимает вдвое больше места, чем переменная типа `int`, но зато может хранить куда большие числа: от -2147483648 до 2147483647 ;

`float` — значениями переменных этого типа являются действительные (они же вещественные) числа.

2. По-другому записывается команда присваивания — вместо « $=$ » пишется просто знак равенства. Так что не надо считать, что запись `a = a + 1` в программе на Си противоречит математическим законам: вставьте мысленно (но только мысленно!) привычное двоеточие, и Вы все поймете.

3. Посмотрим теперь на команду `printf`. Вы уже использовали ее при переводе предыдущей программы для печати строки. Пришла пора рассказать о ней подробнее. Как уже известно, по команде `printf` на экране печатается та строка, которая заключена в двойные кавычки и написана на месте первого параметра этой команды. На самом деле команда устроена несколько хитрее. Именно, не каждый символ заданной ей строки она дословно печатает на терминале. Помимо обычных печатаемых символов прямо внутри строки могут содержаться указания вида «а в этом месте надо напечатать число». Такие указания начинаются символом процента и содержат, помимо этого символа, по крайней мере еще одну букву. Вот какие указания возможны:

- `%d` — напечатать целое число,
- `%u` — напечатать целое положительное число,
- `%c` — напечатать символ,
- `%g` — напечатать вещественное число,
- `%lg` — напечатать длинное целое число.

Если команда `printf` встречает в печатаемой строке такие символы, она выводит на экран не их, а соответствующий им объект (символ или число). Откуда же берутся эти символы и числа? Команда `printf` берет их по очереди из параметров, написанных после печатаемой строки. Рассмотрим подробнее,

как будет работать первая команда печати в нашей программе. Поначалу все «обыкновенные» символы строки «До обмена $A = \%d$ $B = \%d$ » будут выводиться на экран. Так будет продолжаться, пока дело не дойдет до первого символа «%», т. е. кусок строки «До обмена $A =$ » будет напечатан обычным образом. Затем команда возьмет значение первого параметра после строки, т. е. значение переменной a , и выведет его на экран, после чего вывод «обыкновенных» символов строки продолжится. Так на экране появится число 2, а вслед за ним строка « $B =$ ». Заметьте, что пробел в этом куске строки важен — иначе буква B печаталась бы сразу после цифры 2 и текст стал бы неудобочитаемым. Вторая пара символов « $\%d$ » приведет к выдаче на печать значения следующего параметра команды `printf` — в данном случае значения переменной b . Предупреждаем: ни компилятор, ни команда `printf` не проверяют соответствия печатаемой строки и параметров после нее, так что ошибка в этом месте будет обнаружена лишь при выполнении программы.

4. Еще одна особенность команды `printf`, которая проиллюстрирована при втором ее употреблении. В Си нет специальной команды для перевода строки. Предполагается, что пользователь должен переводить строку, посылая на экран соответствующий управляющий символ. Поясним, что это такое. Некоторые символы при выводе их на печать не появляются на экране, а используются в качестве управляющих. Вот номера управляющих символов, допустимых на рассматриваемом нами ПК при работе на любом языке, в том числе и на Си:

10 — вывод символа с этим номером на печать приводит к переходу на следующую строку (с роллированием, если оно требуется);

7 — подать звуковой сигнал;

13 — переставить курсор в самую левую позицию текущей строки;

8 — сдвинуть курсор на одну позицию влево.

По целому ряду причин присутствие символов с такими кодами в текстовом файле с программой нежелательно, даже если они употреблены внутри строки. В Си для набора этих (и некоторых других) символов зарезервировано несколько двухсимвольных сочетаний, начинающихся с символа «обратная косая черта». Вот некоторые из них:

`\n` — символ с кодом 10, т. е. команда перевода строки при печати;

`\7` — символ с кодом 7 («сигнал»);

`\r` — символ с кодом 13 (возврат каретки);

`\b` — символ с кодом 8 (возврат на шаг);

`\\` — обратная косая черта (один символ!);

`\f` — символ с кодом 12. Этот символ приводит к переходу на следующую страницу при печати строки на принтере. При

выводе на дисплей он воспроизводится на экране как обычный символ.

Теперь вы понимаете, куда делась команда перевода строки перед второй печатью: она превратилась в управляющий символ `\n` в самом начале печатаемой строки. Такой символ может встречаться и внутри строки, т. е. одна команда печати способна выводить тексты из нескольких строк.

Упражнение. Придумайте иной способ расположения на экране информации, выводимой последней программой, и измените программу так, чтобы она это расположение обеспечила.

Команды ввода. Новая команда, которая появляется в программе 3, — команда получения символа с клавиатуры. Посмотрим, как выглядит эта команда в программе на Си:

```
main () {  
    unsigned char a;  
    a=getch ();  
    printf («%c%c», a, a);  
}
```

В левой части оператора присваивания записывается имя символьной переменной, в которую мы хотели бы поместить номер введенного с клавиатуры символа. В правой части записано арифметическое выражение, состоящее из обращения к функции `getch`. У нее нет аргументов, но пару скобок после имени необходимо писать, чтобы компилятор мог отличить функцию от переменной. Стандартная функция `getch` устроена так, что при необходимости ее вычисления программа сначала ждет, пока на клавиатуре не наберут какой-нибудь символ, а затем возвращает целое (т. е. типа `int`) число — номер набранного символа. Правила языка Си позволяют употреблять функцию `getch`, как и любую другую функцию, в составе произвольного арифметического выражения, но мы рекомендуем рассматривать строку «`=getch();`» как неделимое ключевое слово, слева от которого нужно написать имя переменной, в которую мы хотим ввести символ, и употреблять эту функцию только в таком контексте.

Убедитесь, что при запуске этой программы компьютер перестает подавать какие-либо признаки жизни и он будет находиться в таком состоянии до тех пор, пока Вы не прикоснетесь к клавиатуре. Заметьте: все, что будет выведено после этого на экран (кстати, что именно?), — это работа функции `printf`, команда ввода символа ничего не делает с экраном!

Команда «ввести» называется на Си `scanf`. Мы не будем описывать ее подробно, а лишь покажем, как перевести на Си команды ввода переменных, имеющих стандартные типы:

Описание переменной	Команда ввода	Перевод на Си
<code>int k;</code>	ввести (k)	<code>scanf ("%d", &k);</code>
<code>unsigned int r;</code>	ввести (r)	<code>scanf ("%u", &r);</code>
<code>unsigned char a;</code>	ввести (a)	<code>scanf ("%c", &a);</code>
<code>long int l;</code>	ввести (l)	<code>scanf ("%ld", &l);</code>
<code>float f;</code>	ввести (f)	<code>scanf ("%g", &f);</code>

(Внимание: пробел перед «%с» при вводе символа важен! У нас нет возможности объяснить причину, но поверьте, что это так.)

В качестве примера использования команды `scanf` рассмотрим перевод на Си программы 4:

```
main () {
    int x;
    printf («\nВведите целое число»);
    scanf ("%d", &x);
    printf («Квадрат числа %d равен %d», x, x*x);
}
```

Параметрами команды `printf`, как Вы помните, могут быть произвольные арифметические выражения. Довольно простое выражение, обозначающее квадрат значения переменной `x`, мы видим среди параметров `printf` в данной программе. Стоит, видимо, рассказать о правилах записи арифметических выражений на Си чуть подробнее.

Арифметические выражения в Си записываются и вычисляются по обычным правилам: скажем, умножение выполняется раньше сложения, могут быть использованы круглые скобки и т. п. Целые числа (любого типа) можно складывать, вычитать, умножать (обозначения «+», «-», «*»). Допускается целочисленное деление, а именно, запись «`n / d`» означает частное, а запись «`n % d`» — остаток от деления нацело `n` на `d`. (Внимание: лишь при делении положительных чисел значение остатка соответствует математическому определению!) Над символами допускаются те же операции, что и над числами — ведь символичные переменные фактически хранят номер некоторого символа, т. е. целое число. Переменные действительного типа можно складывать, вычитать, умножать, делить (обозначения те же). Примеры выражений:

```
(alpha + phi) / gamma * 23.5e2
(k + 13) % 7
```

Предполагается, что переменные, использованные в первом выражении, описаны как действительные, а переменная `k` — целая. Если операция производится над значениями разных типов, одно из значений преобразуется к типу другого. Мы не

будем описывать принятые в Си правила преобразования, а договоримся не смешивать типы в одном выражении.

Еще один класс операций, допустимых над целыми и действительными числами,— это операции сравнения. Вот они:

$=$ равно, $!=$ не равно,
 $>$ больше, $>=$ больше или равно,
 $<$ меньше, $<=$ меньше или равно.

Результатом такой операции, как мы помним, должно быть одно из двух специальных значений («да»/«нет», «истина»/«ложь» и т. п.). В Си такие значения специально не определены, и в результате сравнений получаются целые числа: единица, если соответствующее условие выполнено, и ноль в противном случае. В принципе можно с ними обращаться, как с обычными числами, т. е. выражение вроде « $(b > c) + 12$ » не противоречит правилам Си и равно 13, если b больше c , и 12 в противном случае. Мы, однако, рекомендуем использовать операции сравнения лишь там, где их принято использовать — в конструкциях выбора и цикла. Кстати, пришла пора рассказать о том, как же кодируются эти конструкции на Си.

Программные конструкции. В следующей нашей программе — программе 5 — встречается программная конструкция выбора. Посмотрите, как она выглядит на Си:

```
main(){
    int a, b, max;
    printf («\nВведите два целых числа:»);
    scanf («%d», &a); scanf («%d», &b);
    if (a > b){
        max = a;
    }else{
        max = b;
    }
    printf («\nМаксимальное из введенных чисел — %d»,
        max);
}
```

Проиллюстрируем общую схему перевода конструкции выбора на язык Си. Слева дана конструкция выбора на нашем учебном языке, справа — перевод ее на Си:

ЕСЛИ условие ТО	if (условие){
действие A1	действие A1;
действие A2	действие A2;
действие A3	действие A3;

ИНАЧЕ	}else{	...
действие B1	действие B1;	
действие B2	действие B2;	
действие B3	действие B3;	

КОНЕЦ ЕСЛИ)

Предполагается, естественно, что соответствующие действия и условия также должны быть переведены на Си.

Следующая из построенных нами программ иллюстрирует применения конструкции цикла. Вот как выглядит программа 6, если ее записать на Си:

```
main(){
    int x, max;
    /* последнее и максимальное из введенных чисел,
                                   соответственно */
    unsigned char answer; /* ответ: симв */
    /* принимает значение «п», если ввод чисел не закон-
                                   чен */
    printf («\nВведите число:»);
    scanf («%d», &x);
    printf («\nЭто все? (y/n)»);
    scanf («%c», &answer);
    max=x;
    /* max=максимальное из всех введенных чисел */
    /* ответ=«п», если есть еще числа */
    while (answer == 'n'){
        /* max=максимальное из всех введенных чисел
                                   */
        /* есть еще числа */
        printf («\nВведите число:»);
        scanf («%d», &x);
        printf («\nЭто все? (y/n)»);
        scanf («%c», &answer);
        /* max=максимальное из всех введенных чисел,
                                   не считая последнего */
        /* x=последнее введенное число */
        if (max<x){
            max=x;
        }
        /* max=максимальное из всех введенных чисел
                                   */
        /* ответ=«п», если есть еще числа */
    }
    /* max=максимальное из всех введенных чисел */
    /* чисел больше нет, т. е. max=максим. из всех */
    printf («\nМаксимальное из введенных чисел — %d»,
                                   max);
}
```

Таким образом, конструкция цикла переводится на Си по такой схеме:

ПОКА условие	ПОВТОРЯТЬ while (условие){
действие 1	действие 1;
действие 2	действие 2;
действие 3	действие 3;
КОНЕЦ ПОКА	}

Вы уже заметили, что фигурные скобки исполняют на Си роль ограничителей в программных конструкциях. Для выделения комментария используются два двухсимвольных сочетания: «/*» в начале комментария и «*/» в конце. Еще раз обратите внимание на операцию «равно» — она обязательно должна содержать два знака равенства!

Процедуры. Организация процедур в Си имеет ряд особенностей по сравнению с тем способом работы, о котором мы говорили в гл. 4. Тем не менее, дословный перевод использованных нами процедур на язык Си возможен, хотя выглядеть он будет несколько необычно. Сейчас мы покажем, как делается такой перевод, не поясняя, однако, почему процедуры на Си устроены именно так (про это Вы можете прочитать в любом руководстве по языку). Запишем полный текст программы вычисления максимума с использованием процедур. Вот что у нас получится:

```
input number (number, letter)
int * number; char * letter;
{
    printf («\nВведите число:»);
    scanf («%d», &* number);
    printf («\nЭто все? (y/n)»);
    scanf («%c», &* letter);
}
main(){
    int x, max;
    unsigned char answer;
    input_number (&max, &answer);
    while (answer == 'n'){
        input_number (&x, &answer);
        if (max < x){
            max = x;
        }
    }
    printf («\nМаксимальное из введенных чисел — %d»,
        max);
}
```

Посмотрим сначала на текст процедуры, который должен предшествовать тексту основной программы. Наверняка Вы сами разберетесь с мелкими деталями перевода, вроде того, что слово «процедура» отсутствует, а сразу идет имя процедуры, а формальные параметры описываются не внутри списка, а после заголовка (но до первой фигурной скобки!). Странность заключена в способе описания и использования формальных параметров, и ее можно пояснить таким прагматическим правилом:

при описании выходных формальных параметров вместо слова «вых» пишется звездочка;

при использовании выходных формальных параметров внутри описания процедуры они всякий раз сопровождаются звездочкой, которая должна предшествовать имени параметра.

Теперь займемся текстом основной программы и посмотрим, как происходит обращение к такой процедуре. Здесь есть лишь одно отличие — при обращении к процедуре на месте выходного параметра должно быть по-прежнему написано имя переменной, но только перед этим именем должен еще стоять знак «&» (его называют «коммерческое и» или амперсанд). В отношении входных параметров никаких специальных правил нет, так что процедура вычисления гипотенузы будет выглядеть на Си так:

```
calcul_gip (k1, k2, gip)
float k1, k2, *gip;
{
    *gip = sqrt (k1*k1 + k2*k2);
}
```

Здесь sqrt — имя стандартной функции Си, которая вычисляет квадратный корень. Оформление этой процедуры в виде функции будет выглядеть так:

```
float calcul_gip (k1, k2)
float k1, k2;
{
    return (sqrt (k1*k1 + k2*k2));
}
```

Как видим, здесь нет никаких сложностей, только вместо слова «ответ» пишется слово return и функция описывается в традициях Си — сначала имя типа, а потом описываемый объект.

Заметим, что правило записи выходных параметров соблюдается и при вызове стандартных процедур Си, каковыми являются, например, printf и scanf. Так, амперсанд возле второго параметра при вызове scanf нужен именно потому, что этот параметр выходной — он получает новое значение в результате вызова scanf. Этого нет в команде printf, поскольку все ее параметры — входные, они лишь информируют компьютер о том, что именно надо напечатать, сами при этом не меняясь.

На этом мы закончим введение в язык Си. Если этот язык Вас заинтересовал и есть желание изучить его подробнее, вооружитесь любой книгой с описанием этого языка, хотя бы одной из тех, что указаны в списке литературы в конце этой главы. Мы же в заключение дадим начинающему несколько советов по работе на Си.

Будьте внимательны! Это может показаться банальным, но все же мы повторим: будьте предельно внимательны при работе на Си! Дело в том, что язык спроектирован так, что многие ошибки компилятором не распознаются, а проявляются только

во время выполнения уже готовой программы и понять, в чем дело, бывает не так-то просто. Поэтому

1) правильно записывайте операцию «равно»! Если написать «if (a = b)...», компилятор не скажет ничего, но программа будет выполняться совершенно иначе, чем при написании «if (a == b)»;

2) не забывайте ставить амперсанды и звездочки при передаче и, соответственно, использовании выходных параметров. Такие ошибки тоже проявляются лишь на стадии выполнения. Например, запись «scanf («%d», x)» в некоторых случаях может привести к зависанию программы;

3) у Вас есть возможность вставить в ваш файл с программой на Си описания правил использования стандартных процедур. Эти описания содержатся в файлах с расширением H в каталоге \CC\INCLUDE. В частности, файл с именем STDIO.H относится к процедурам ввода и вывода. Вы можете приказать компилятору добавить этот файл к Вашей программе, поместив в начале Вашей программы строку

```
#include <stdio.h>
```

Такое добавление приведет к тому, что компилятор начнет замечать некоторые ошибки при вызовах стандартных процедур (правда, не все).

Оставшуюся часть параграфа мы посвятим рассказу о том, как записать на Си вызовы специфичных для ПК процедур, которые мы описали в предыдущей главе. Фактически он содержит справочный материал и ориентирован на пользователя, имеющего опыт работы на Си.

Управление ОС и устройствами. Ниже мы переведем на СИ команды, о которых шла речь в предыдущей главе, сохраняя ту же последовательность. В некоторых случаях приводится образец обращения к эквивалентной стандартной команде (т. е. к процедуре, описание которой имеется в стандартной библиотеке каждого из описанных нами компиляторов), в других — фрагмент программы, приводящий к тому же результату, что и переводимая команда, в третьих — мы приведем полный текст описания соответствующей процедуры на Си. Если перевод какой-либо команды никак не прокомментирован, это означает, что она выполняется точно так же, как описано в гл. 5. Прием еще несколько соглашений.

1. В примерах команд будем для наглядности использовать переменные с русскими именами, хотя Ваш компилятор этого, скорее всего, не допускает.

2. Если тип переменной не прокомментирован, то подразумевается тот тип, что был указан в предыдущей главе. Исключение составляют переменные, про которые мы писали «лог», так как в Си нет такого типа. Во всех переводимых дословно командах, в которых указан тип «лог», в Си должен быть тип int с обычным соглашением — нулевое значение кодирует ЛОЖЬ, любое ненулевое рассматривается как ИСТИНА.

3. Перед командой может быть написано несколько дополнительных строчек, как правило состоящих из инструкций include. Это означает, что для использования описываемой команды так, как предлагаем ее использовать мы, следует добавить эти строки в начало Вашей программы.

4. Параметром многих команд является строчка с именем файла. Путь к файлу может быть указан по обычным правилам ОС, но надо учитывать, что символ «обратная косая черта» кодируется на Си двусимвольной конструк-

цией «\». В то же время ОС позволяет в данном случае употреблять вместо обратной косой черты обычную, т. е. путь «C:\MYDIR\MYFILE. EXT» должен быть записан в программе на Си в виде строки «C:\\MYDIR\\MYFILE. EXT» либо «C:/MYDIR/MYFILE. EXT».

5. Ряд команд на Си записывается в виде
ошибка = имя _ команды (параметры)

Смысл переменной слева от знака равенства в данном случае противоположен смыслу переменной «удалось», т. е. она становится равной нулю, если команда выполнена успешно, и отличной от нуля (как правило, равной минус единице) в противном случае.

В стандартных библиотеках языка Си нет некоторых специальных команд. Учитывая, что эти команды все-таки очень полезны, мы написали свою небольшую библиотеку процедур работы с Си, которая обеспечивает выполнение описанных нами команд. Эта библиотека находится в файлах file. c, find. c, low. c, а описания процедур — в файле user. h. Тексты этих файлов приведены в приложении. Для того чтобы можно было работать с описанными нами командами, требуется ввести эти тексты в ПК и откомпилировать их. В Ваши программы должна быть добавлена инструкция # include «user. h», а инструкция на изготовление выполняемого файла должна выглядеть так:

LINK MYPROG + имя объектного файла, в котором содержится описание используемой процедуры:

Все приведенные нами команды могут быть выполнены при работе с любым описанным компилятором. Отличие заключается, в основном, в названиях H-файлов. Кроме того, компилятор Турбо имеет несколько более широкий набор библиотечных подпрограмм, так что при работе на нем часть файла user. h не требуется. Мы указываем отличия при описании каждой из команд, ставя букву T в квадратных скобках возле тех строк, которые нужны только при компиляции с помощью Турбо-Си, и букву M, если эта строка требуется лишь компилятору Microsoft.

Итак, начнем с команд работы с файлами. Открывается файл командой:

```
# include <stdio. h>
```

код файла = fopen («имя файла», зачем);

второй параметр которой может быть записан, в частности, так:

«rb»—ДЛЯ ЧТЕНИЯ,

«wb»—ДЛЯ ЗАПИСИ,

«ab»—ДЛЯ ДОБАВЛЕНИЯ.

Код файла—переменная специального типа FILE*. Если файл открыть не удалось, то в качестве кода файла возвращается специальная константа NULL. С открытым таким образом файлом можно работать с помощью таких команд:

```
# include <stdio. h>
```

прочтено = fread (куда, размер, сколько, кодфайла);

записано = fwrite (откуда, размер, сколько, кодфайла);

Первый параметр обеих команд должен иметь тип void* и задавать адрес памяти, куда (или откуда) должна производиться пересылка информации. Обычно на месте этого параметра пишется имя описанного в программе символического массива. Третий параметр задает число блоков информации, которое надо прочитать или записать, а второй параметр—размер одного блока в байтах (оба параметра типа int). Результатом обеих функций является число блоков, которое удалось прочитать или записать. Если это число оказалось меньше, чем заданное третьим параметром, значит, полностью выполнить команду не удалось (читаемый файл кончился, нет больше места на диске и т. п.).

Закрывается файл так:

```
# include <stdio. h>
```

ошибка = fclose (кодфайла);

где переменная «ошибка» получает нулевое значение в случае, если это удалось.

Обмен информацией с файлом может производиться и с помощью команд, аналогичных printf и scanf. Соответствующие команды называются

fprintf и fscanf, а код файла пишется в начале списка параметров. Пример:

```
# include <stdio. h>
FILE* code; float x;
code = fopen («RESULT. DAT», «wb»);
fprintf (code, «Вычисленный результат равен %g», x);
```

Перейдем теперь к командам обработки информации о файлах. К сожалению, не существует стандартных функций Си, сообщающих о виде файла в той форме, в которой он был нами определен, так что нам пришлось описать приведенные ниже процедуры самостоятельно. Соответствующие описания приведены в конце этого параграфа, а о том, как обращаться к этим процедурам, мы сейчас расскажем. Выражение целого типа

```
# include «user. h»
(file __ attr («имя файла») & вопрос)
возвращает ответ на соответствующий вопрос, который может быть записан так (см. гл. 5):
```

```
FA _ DIREC — КАТАЛОГ,
FA _ RDONLY — ЗАЩИЩЕН,
FA _ HIDDEN — СКРЫТЫЙ,
FA _ ARCH — АРХИВНЫЙ,
FA _ SYSTEM — СИСТЕМНЫЙ,
FA _ LABEL — МЕТКА ТОМА.
```

Значение выражения будет равно либо единице, что значит «да», либо нулю, что означает «нет». Пример:

```
if (file __ attr («myfile. ext») & FA _ RDONLY) {
    printf («Файл защищен от изменения»);
}
```

Заметим, что данное обращение может быть записано и еще одним способом:

```
# include <stdio. h>
if (! access («myfile. ext», 2)) {
    printf («Файл защищен от изменения»);
}
```

Функция access стандартная, но с ее помощью можно получить ответ лишь на этот вопрос.

Следующие две функции возвращают результат типа long — размер файла и время его последней модификации:

```
# include «user. h»
size __ of __ file («имя файла»)
time __ of __ file («имя файла»)
```

Время исчисляется довольно экстравагантно—как число секунд, прошедшее с 1 января 1970 г. Позднее мы расскажем, как извлечь из этого числа дату и время в обычной форме. А изменить время модификации файла можно, выполнив такие две команды:

```
# include <types. h>
# include <utime. h>
struct utimbuf буфер;
буфер. modtime = время;
ошибка = utime («имя файла», & буфер);
```

В результате время модификации файла устанавливается равным тому значению, что хранится в переменной «время» типа long и трактуется так же, как мы только что объяснили. Если на месте второго параметра стоит специальная константа NULL, то время модификации устанавливается равным текущему времени по часам ОС.

Команду «изменить вид файла» мы перевели так:

```
# include «user. h»
set __ file __ attr («имя файла», вопрос, ответ, & удалось);
Вопрос записывается так же, как и в функции file __ attr, например:
set __ file __ attr («myfile. ext», FA _ RDONLY, 0, & done);
set __ file __ attr («myfile. ext», FA _ RDONLY, 1, & done);
```

Два приведенные примера могут быть записаны лишь с использованием стандартных процедур, соответственно

```
# include <types.h>
# include <stat.h>
# include <io.h>
done = ! chmod («myfile. ext.», S_IWRITE);
done = ! chmod («myfile. ext.», S_IREAD);
```

К сожалению, процедура chmod может менять только признак FA_RDONLY-NLY. Проверить, существует ли файл с заданным именем, можно при помощи функции

```
# include <stdio.h>
access («имя файла», 0)
```

которая возвращает ноль только в том случае, если файл с указанным именем существует. Следующие три команды осуществляют поиск по каталогу;

```
# include «user.h»
start __ find («образец», вид, &удалось);
is __ next __ file ()
next __ file (имя);
```

Поскольку описания этих трех команд мы написали сами, перевод получился дословный. Заметим лишь, что параметром последней команды должно быть имя символьного массива, связка «или» в записи вида файла заменяется знаком «|» (побитовое или). Иными словами, приведенный в предыдущей главе пример можно перевести так:

```
start __ find («c:\mydir\*. *», FA_HIDDEN | FA_SYSTEM, & done);
```

Для поиска лишь среди обычных файлов следует написать на месте второго параметра этой команды ноль.

Команды переименования и уничтожения файла соответственно выглядят на Си так:

```
# include <stdio.h>
ошибка = rename («старое имя», «новое имя»);
ошибка = remove («имя файла»);
```

Похожим образом переводятся команды создания и уничтожения каталога:

```
# include <direct.h> [M]
# include <dir.h> [T]
ошибка = mkdir («путь»);
ошибка = rmdir («путь»);
```

Номер ошибки, если она была совершена, можно выяснить так:

```
# include <stdlib.h>
причина = __ doserrno;
```

Как видим, номер ошибки хранится в специальной переменной.

Так выглядят функция запроса и команда смены текущего диска:

```
# include «user.h» [M]
# include <dir.h> [T]
getdisk ()
setdisk (номер);
```

А так—функция запроса и команда смены текущего каталога:

```
# include <direct.h> [M]
# include <dir.h> [T]
удалась = (getcwd (имя, размер) != NULL);
ошибка = chdir («путь»);
```

Первая из приведенных выше команд помещает строку, содержащую путь к текущему каталогу в символьный массив; имя которого написано на месте первого параметра функции getcwd. При этом предполагается, что следующий параметр задает размер этого массива, так что если нужная строка длиннее, то команда оканчивается неудачно. Вторая команда позволяет сделать текущим каталог с заданным именем и не требует комментариев.

Команды «выход» и «система» переводятся на Си следующим образом:

```
exit (код_завершения);  
ошибка = system («текст команды»);
```

Перейдем к командам работы с портами и памятью. Вот как выглядит обмен с портами на Си:

```
# include <conio. h> [M]  
# include <dos. h> [T]  
куда = inr (какой); /* прочесть порт */  
outp (куда, что); /* записать порт */
```

Напомним, что в Си есть возможность записывать числа в 16-ричной системе счисления, добавляя к ним спереди цифру 0 и букву х, например 61Н запишется как 0x61, так что есть возможность записывать адрес порта в том виде, в каком Вам удобнее это делать. В частности, следующие четыре строки с точки зрения Си эквивалентны:

```
outp (0x61, 0x4B);  
outp ( 91, 0x4b);  
outp (0x61, 75);  
outp ( 97, 75);
```

Команды записи в память и чтения по абсолютным адресам памяти можно записать так:

```
# include «user. h» [M]  
# include <dos. h> [T]  
  
peekb (база, смещение) /* байт памяти */  
pokeb (база, смещение, что); /* записать байт */  
peek (база, смещение) /* слово памяти */  
poke (база, смещение, что); /* записать слово */
```

На самом деле эти команды реализованы в виде макроопределений. Если Вы имеете опыт программирования макро на Си, то, наверное, и сами сможете реализовать подобные команды. Параметры «база» и «смещение» должны иметь целый тип. Поскольку нельзя просто преобразовать в адрес целое число и наоборот, полезными окажутся функции, разбирающие адрес на части и собирающие его из двух частей. Вот они:

```
# include «user. h» [M]  
# include <dos. h>  
FP _ SEG (адрес) /* возвращает базовую часть указателя */  
FP _ OFF (адрес) /* возвращает смещение */  
MK _ FP (база, смещение) /* возвращает адрес, т. е. значение типа  
«указатель» */
```

Команда, пересылающая участки памяти, переводится дословно:
metcpu (куда: адрес, откуда: адрес, сколько: цел)

Заметьте, что первые два параметра должны быть адресами, т. е. это или обычные указатели Си, или имена массивов, или адреса, собранные при помощи функции MK _ FP.

Перейдем к командам работы с клавиатурой. Все они, как мы помним, имели вид функций без аргументов. Вот их список:

```
# include <dos. h> [T]  
# include «uiser. h»  
  
kbhit () /* есть символы */  
k _ ascii () /* ASCII _ КОД */  
k _ extc () /* расш _ код */  
k _ pop (); /* убрать _ символ */  
k _ flush (); /* опустошить */  
k _ regs () /* регистры */
```

Как видим, перевод дословный, поскольку описания этих функций мы писали самостоятельно. Так же обстоит дело и со средствами работы с экраном. Вот как выглядит их набор на Си:

```
#include <dos.h> [T]
#include «user.h»
s __mode () /* режим */
s __setmode (какой); /* задать __режим */
s __page () /* страница __экрана */
s __setpage (какую); /* задать страницу */
s __off (); /* включить __экран */
s __on (); /* выключить экран */
s __poscur (строка, столбец); /* поз __курсор */
s __curline () /* строка __курсора */
s __curcol () /* столбец __курсора */
s __drawpoint (строка, столбец, цвет);
/* покрасить __точку */
s __point (строка, столбец) /* цвет __точки */
s __setback (какой); /* задать цвет фона */
s __setpal (какую); /* задать палитру */
```

Массивы VM0, VM1 и т. д. описаны в файле USER. Н и могут быть использованы в точности так же, как мы это делали в гл. 5. В качестве примера переведем на Си программу горизонтального роллирования экрана из § 5.4:

```
#include «user.h»
main () {
    int moved, line;
    unsigned char color, letter;
    moved = 0;
    while (moved < 80) {
        line = 0;
        while (line < 25) {
            color = VM0 [line, 79]. a;
            letter = VM0 [line, 79]. c;
            memcpy (&VM0 [line, 1], &VM0 [line, 0], 79);
            VM0 [line, 0]. a = color;
            VM0 [line, 0]. c = letter;
            line = line + 1;
        }
        moved = moved + 1;
    }
}
```

С помощью команд работы с экраном несложно написать программу, которая создает на экране красивые разноцветные иерархические наборы меню, управлять которыми можно с помощью курсора. Пример таких меню приведен на рис. 6.1, а программа, их генерирующая, приводится в файле DIAL. С в приложении.

Опишем теперь пару команд работы с системными часами (таймером). В конкретных версиях Си могут быть и другие команды для сходных целей. Они могут оказаться удобнее описанных нами, но их наличие и вид зависят от конкретного компилятора. Команда

```
#include <time.h>
time (& время)
```

помещает в переданную ей переменную (типа long int) число секунд, прошедшее с начала 1970 г. Это число может быть «разобрано» на привычные нам элементы при помощи команды

```
#include <time.h>
часть = localtime (& время);
```

где время задано как и в команде time, а переменная «часть» должна иметь специальный тип struct tm *. После выполнения данной команды эта пере-

мённая указывает на структуру, содержащую «расшифрованную» информацию о времени. Перечислим компоненты этой структуры (все они типа int):

```
часть —> tm __ sec    / * секунды */  
часть —> tm __ min    / * минуты */  
часть —> tm __ hour   / * часы */  
часть —> tm __ mday   / * число */  
часть —> tm __ mon    / * месяц: 0—Январь и т.д. */  
часть —> tm __ year   / * год минус 1900 */  
часть —> tm __ wday   / * день недели: 0—Воскресенье и т. д. */  
часть —> tm __ yday   / * день года: 0—1 января и т. д. */
```

Приведем пример программы, выводщей на экран текущую дату по часам ОС:

```
#include <stdio.h>  
#include <time.h>  
main () {  
    struct tm * part; long int t;  
    time (&t);  
    p = localtime (&t);  
    printf («Дата — %d. %d. %d\n», p —> tm __ mday, p —> tm __  
mon, p —> tm __ year);  
}
```

В заключение перечислим команды управления принтером и динамиком:

```
#include <dos.h> [T]  
#include «user.h»  
  
lpt __ begin (номер); / * начать __ работу */  
lpt __ status (номер) / * состояние __ принтера */  
Ответ = lpt __ send (номер, что); / * передать __ принтеру */  
speaker __ on (); / * включить __ динамик */  
speaker __ off (); / * выключить __ динамик */  
oscill __ on (); / * включить __ генератор */  
oscill __ off (); / * выключить __ генератор */  
set __ period (i) / * задать __ период */
```

6.3. ЯЗЫК БЕЙСИК

Запуск программы на языке Бейсик. Что необходимо сделать пользователю для того, чтобы выполнить программу на Бейсике? Для ПЭВМ, совместной с моделью IBM PC XT, разработано большое число различных компиляторов и интерпретаторов языка Бейсик.

Интерпретатор Бейсика с графическими возможностями записан на отдельной дискете и называется BASICA, или BASICA, или GW-BASIC. Для его установки в ПЭВМ достаточно скопировать файл с одним из указанных имен на фиксированный диск или, если в компьютере установлены только флоппи-диски, просто обратиться к этой программе с дискеты, предварительно сделав копию этого файла. Итак, наберем на клавиатуре BASICA и дадим возврат каретки. При этом экран очистится, в верхней его части появится «товарный знак» — версия и производитель интерпретатора и сочетание «ок». Непосредственно под словом «ок» будет мигать курсор. В этом режиме интерпретатор ждет ввода команд с клавиатуры. Обратим внимание на нижнюю часть экрана. В ней

появилась строка меню, в которой содержатся названия функциональных клавиш F1 — F10 и набор функций, выполняемых при нажатии на эти клавиши. Для знакомства с интерпретатором нам будут нужны только первые четыре клавиши следующего содержания:

F1 LIST — выдача текста набранной программы,
F2 RUN — запуск программы,
F3 LOAD" — чтение файла с текстом программы с диска,
F4 SAVE" — запоминание текста в файле на диске.

В интерпретаторе Бейсика мы можем сразу же начать вводить текст программы. Переведем на Бейсик первую из написанных в гл. 4 программ. На Бейсике эта программа состоит всего из двух строчек и записывается совершенно естественным образом:

```
10 PRINT "Программа 1 выполнена!"  
20 END
```

Заметим, что в Бейсике отсутствует присвоение имени программы и обозначение ее начала, но зато имеется обозначение ее конца (оператор END). Начнем набирать ее, как будто мы находимся в обычном текстовом редакторе. Единственное отличие интерпретатора Бейсика от текстового редактора состоит в том, что для фиксации изменений в каждой строке программы необходимо дать возврат каретки (при этом строка не разрывается). В остальном же изменение текста ничем не отличается от работы с экранным редактором: мы можем передвигаться по экрану в пределах видимого окна, вставлять новые символы, стирать старые и т. п. Вывод текста на экран производится нажатием на клавишу F1. Если нам необходимо вывести только часть текста программы, можно дать эту команду с указанием номеров строк, например команда LIST 140—200 выведет на экран строки с номерами от 140 до 200. Набрав текст программы, нажмем на функциональную клавишу F2. Предположим, что мы допустили синтаксическую ошибку: в строке 10 написали PRIT вместо PRINT. Что произойдет в этом случае? Раздастся звуковой сигнал и на экране появится надпись:

```
Syntax error in 10  
10 PRIT "Программа 1 выполнена!"
```

При этом курсор будет стоять под цифрой 1 номера строки. Это сообщение означает, что интерпретатор нашел синтаксическую ошибку. Теперь мы можем сразу же начать редактирование неправильной строки. Для этого с помощью клавиш управления курсором подведем его к букве T, нажмем на клавишу Ins и напечатаем букву N. Теперь строка будет выглядеть правильно. Нажмем возврат каретки (зафиксируем изменение в памяти) и снова нажмем на клавишу F2. Программа выполнится и на экране появится надпись

Программа 1 выполнена!
Ok

Теперь можно редактировать программу и запускать ее новые варианты. Для того чтобы запомнить текст на диске, нажмем клавишу F4. На экране появится надпись SAVE. Допишем имя файла (назовем его PROG1) и дадим возврат каретки. Для считывания файла проведем аналогичную операцию, нажав на клавишу F3. Заметим, что если мы хотим использовать подготовленную программу на Бейсике для последующей компиляции, то для записи программы в файл следует дать команду

SAVE "PROG1", A

Для выхода из интерпретатора наберем на клавиатуре system и дадим возврат каретки, после чего окажемся в ДОС.

Как видим, работа с интерпретатором крайне проста. Мы уже неоднократно рассматривали различия между компиляторами и интерпретаторами. Для разработки серьезных и больших программ на Бейсике имеет прямой смысл пользоваться компилятором: программа работает быстрее, ее можно разбить на несколько программных модулей. При разработке коротких и простых программ имеет смысл пользоваться интерпретатором Бейсика, так как процесс отладки программы в нем гораздо эффективнее.

Рассмотрим работу с компилятором Бейсика. Для ПЭВМ существует несколько компиляторов: Professional BASIC, Quick Basic, IBM Basic, а также редактор-компилятор Turbo-Basic. Редактор-компилятор Бейсика очень похож на описанный в § 6.1 редактор-компилятор Турбо-Си: он также сочетает в себе свойства редактора программ, компилятора и отладчика и очень удобен в работе. Компилятор Бейсика фирмы IBM, версия 2, записан на двух гибких дисках. На одном из них записаны библиотеки для так называемых ISAM-файлов, которые не понадобятся пользователю, если только он не захочет создавать систему типа базы данных на Бейсике. На другом диске записан собственно компилятор Бейсика, который называется BASCOM. EXE, и библиотека Бейсика BASCOM20. LIB. Кроме того, на диске имеются файлы BASRUN20. LIB — вспомогательная библиотека и BASR — UN20. EXE — специальная программа для текущих применений. Наличие этих двух файлов связано с тем, что стандартные библиотеки Бейсика можно потребовать подгружать непосредственно в момент выполнения программы. Это уменьшает размеры кода программы. Альтернативным является «классическое» решение, при котором все библиотеки подгружаются к программе из файла BASCOM20. LIB в процессе сборки. Для установки компилятора Бейсика в ПЭВМ, вообще говоря, достаточно скопировать эти файлы на фиксированный диск.

Запишем рассмотренную программу в файл с помощью текстового редактора и назовем ее PROG1. BAS. Расширение BAS является стандартным для текстов программ, написанных на Бейсике. Вернувшись из редактора в ДОС, дадим следующую команду:

BASCOM PROG1;

Как для других языков, эта команда означает:

«BASCOM» — откомпилируй,

«PROG1» — программу, находящуюся в этом файле (по умолчанию предполагается, что у файла стандартное расширение .BAS),

«;» — не задавай лишних вопросов.

Проделав все описанное, пользователь увидит на экране «товарный знак» компилятора. Затем компьютер подумает несколько секунд (для маленьких программ в несколько раз меньше, чем при использовании компиляторов с языка СИ) и напишет на экране сообщение о результатах компиляции. Если компилятор обнаружит ошибки в программе, то он сообщит о них. Описываемая версия компилятора Бейсика при этом будет выдавать звуковой сигнал и писать на экране строки программы, в которых обнаружена ошибка с условными сообщениями об этих ошибках. Эти сообщения зависят от версии компилятора. Их описание обычно содержится в руководстве по данной версии языка.

Если компилятору не указать имени исходного файла или не поставить после имени точку с запятой, то он начнет задавать вопросы:

Source filename [.BAS]:

(исходный файл с расширением BAS). Напишем в этой строке PROG1. Расширение здесь писать не нужно. Если на этом этапе после имени файла поставить точку с запятой, то больше отвечать на вопросы не придется, в противном случае компилятор задаст следующий вопрос:

Object filename [PROG1. OBJ]:

Этот вопрос относится к объектному файлу. Компилятор предлагает создать объектный файл с тем же именем и с расширением OBJ. Если это нас устраивает, дадим возврат каретки, в противном случае введем новое имя. Теперь компилятор задаст следующий вопрос:

Source listing [NUL. LST]:

(создавать ли листинг программы). Название файла NUL. LST соответствует тому, что листинг в отдельном файле создан не будет. Если мы дадим возврат каретки, то начнется компиляция. В противном случае необходимо ввести имя файла, в который будет помещен листинг программы. Заметим, что в листинг будут включены все сообщения об ошибках, которые найдет компилятор. Файл листинга можно затем будет просмотреть текстовым редактором.

Итак, если компилятор отработал правильно, на диске появился файл PROG1. OBJ. Теперь необходимо выполнить операцию сборки с помощью линкера. Для этого достаточно написать

```
link prog1;
```

(сделай выполняемую программу prog1. exe не задавая лишних вопросов). Теперь уже можно запустить программу, набрав на клавиатуре PROG1 и нажав на возврат каретки. На экране появится надпись «Программа 1 выполнена!». Будем считать, что мы проверили правильность работы нашего компилятора и теперь можем переходить к разработке более сложных программ на Бейсике.

Заметим, что у компилятора Бейсика имеется ряд дополнительных параметров, задаваемых специальными флажками. Например, если мы напишем компилятору IBM

```
BASCOM PROGRAM / N;
```

то это будет означать: «скомпилируй файл PROGRAM. BAS, в котором мы не проставляли номера строк». Таким образом, программу на Бейсике можно писать и не нумеруя строки.

Краткий обзор языка Бейсик. Теперь, когда мы ознакомились с процессами запуска программ, написанных на Бейсике, сделаем краткий обзор этого языка. Начнем сразу с программы 2 гл. 4. Вот как она выглядит на Бейсике:

```
10 A% = 2 : B% = 3
20 PRINT "До обмена A = "; A%; " B = "; B%
30 X% = A : A% = B% : B% = X%
40 PRINT "После обмена A = "; A%; " B = "; B%;
50 END
```

Поглядев на эту программу, мы видим, что Бейсик допускает запись нескольких операторов в одной и той же строке. При этом последовательные операторы отделяются друг от друга двоеточием. Заметим, что строчку 30 в Бейсике можно переписать следующим образом:

```
30 SWAP A%, B%
```

Команда SWAP автоматически произведет обмен переменных.

Текст этой программы выглядит довольно просто, за исключением того, что после названий переменных почему-то появились обозначения процентов, а в операторе PRINT, который, как мы уже поняли, выводит данные на печать, присутствуют точки с запятыми.

Начнем с описания типов переменных в Бейсике. Сразу отметим необычную форму обозначения различных типов переменных: тип переменной обозначается различными стандартными символами, стоящими после имени переменной. Как мы поняли, целая переменная X должна обозначаться

в программе X%. Какие переменные могут встречаться в программах на Бейсике? Перечислим их:

целые переменные, обозначаемые символом процента «%» и занимающие 2 байта памяти. Диапазон их значений от— 32768 до 32767;

действительные переменные (никак не обозначаются или обозначаются знаком «!»), занимающие 4 байта памяти и имеющие диапазон значений от $-1.701412E+38$ до $1.701412E+38$;

действительные переменные двойной точности, обозначаемые знаком диеза «#», занимающие 8 байт памяти и имеющие шестнадцать значащих цифр (в отличие от 7 значащих цифр у числа одиночной точности);

строковые переменные (обозначаются значком «\$»), занимающие $2+L$ байтов памяти, где L — длина строковой переменной. Эта длина не может превышать 255 знаков.

Приведем несколько примеров присваивания значений переменным:

$A\% = 1$ (целая переменная),

$A = 1.33$ (действительная переменная),

$A\$ = \text{«Это текст»}$ (строковая переменная).

Типы переменных в Бейсике можно определить и иначе. Для этого имеется специальный оператор DEF. Устроен он следующим образом: после оператора указывается тип переменной и буквы алфавита, при этом все переменные, начинающиеся с этих букв, будут по умолчанию иметь заданный тип. Например, оператор DEFINT I, J, K присвоит всем переменным, начинающимся с букв I, J, K, целый тип, а оператор DEFSTR A, C присвоит переменным, начинающимся с A или C строковый тип.

Нам могут встретиться не десятичные, а шестнадцатеричные числа. В Бейсике они записываются следующим образом: перед началом шестнадцатеричного числа ставится знак «&H».

В Бейсике можно задавать массивы любого типа, имеющие до 255 индексов. Определение размерности массива делается с помощью оператора DIM. Например, оператор DIM A%(10, 100) задает целый двумерный массив, имеющий 10×100 элементов. Заметим, что обычно по умолчанию максимальный объем массивов в памяти не может превышать 64 Кбайт. Как же быть, если в компьютере имеется 640 Кбайт памяти и мы хотим использовать очень большие массивы данных? Решить эту задачу можно только с помощью компилятора Бейсика. Для этого в первой строке программы нужно написать специальную метакоманду (без номера строки!):

REM DYNAMIC

Теперь можно смело задать массив большой размерности, «влезавший» в имеющуюся оперативную память. Кстати, объем этой памяти всегда можно посмотреть с помощью

функции FRE. Например, $A = FRE(-1)$ поместит в переменную A число свободных байтов памяти.

Теперь рассмотрим структуру оператора PRINT. Этот оператор предназначен для вывода числовых и текстовых значений и записывается в удобный и совершенно естественной форме, в которой текстовые переменные указываются в кавычках, а переменные обозначаются их именами. Точка с запятой, встречающаяся в операторе PRINT, указывает на то, что печатаемый текст пишется подряд. Так, если в конце оператора стоит точка с запятой, то последующий оператор печати начинается с текущей строки. Именно поэтому в конце первого оператора PRINT мы не поставили точку с запятой (эквивалентно команде «перевести строку»), а в строке 40 поставили ее. Если список вывода переменных в операторе PRINT разделяется запятыми, то каждое следующее числовое значение будет печататься в своей зоне экрана. При этом размер каждой зоны равен 14 позициям. В таком режиме на экране с помощью оператора PRINT можно легко создать подобие таблицы чисел.

К этому моменту мы уже готовы написать на Бейсике программу 3 гл. 4:

```
10 A$ = INPUT$ (1)
20 PRINT A$; A$;
30 END
```

Здесь нам все понятно, кроме оператора INPUT\$. Этот оператор читает один символ, нажатый на клавиатуре, не печатая его на экране. Заметим, что до тех пор, пока пользователь не нажмет на клавиатуре любую клавишу, программа будет ждать. Функция INPUT\$ может прочесть с клавиатуры любое заданное число символов. Так, если в строке 10 написать A = INPUT$ (2)$ и, запустив программу, набрать на клавиатуре «AB», то это сочетание букв попадет в текстовую переменную A\$.

В Бейсике имеется и другая функция, похожая на эту, — INKEY\$. Эта функция читает с клавиатуры только один символ, причем не ждет нажатия клавиши. С использованием команды INKEY\$ наша программа тоже может быть написана, но при этом придется циклически опрашивать клавиатуру до тех пор, пока значение переменной A = INKEY$$ не станет отличным от «нулевой» строки, обозначаемой двойными кавычками " ".

В Бейсике можно также ввести целую строку текста и поместить ее в строковую переменную. Это делается с помощью оператора LINE INPUT. Ввод данных в нем заканчивается после нажатия на клавишу возврата каретки. Например,

```
LINE INPUT "Введите вашу фамилию", NAME$
```

Запишем на Бейсике программу 4 гл. 4:

```
10 PRINT
20 PRINT "Введите целое число";
30 INPUT X%
40 PRINT "Квадрат числа"; X%; "равен"; X% * X%;
50 END
```

Первый оператор PRINT в строке 10 эквивалентен функции перевода строки: он пропустит одну строку на экране. Познакомимся с оператором INPUT. Оператор INPUT позволяет ввести с клавиатуры текст или числовые значения. Этот оператор устроен очень удобно. Например, вместо строк 20 и 30 мы можем просто записать

```
INPUT "Введите целое число"; X%
```

При этом на экране будет напечатано «Введите целое число?» и программа будет ждать до тех пор, пока пользователь не введет число и не даст возврат каретки. Заметим, что в конце текстовой строки автоматически будет помещен вопросительный знак. При желании от него можно избавиться, заменив точку с запятой на запятую. Можно также ввести несколько значений, разделив их запятыми.

Операции ввода и вывода можно поместить в любом месте экрана. Для этого в Бейсике используется оператор

```
LOCATE x, y
```

где x — текущая строка экрана, y — столбец. Например, команда LOCATE 10,5 поместит курсор в пятую позицию десятой строки экрана.

В только что приведенной программе мы впервые столкнулись с операциями над переменными. Как видим, для умножения двух чисел в Бейсике сохранена естественная форма записи. Точно так же операции сложения и вычитания обозначаются соответственно символами «плюс» и «минус», возведение в степень — «крышечкой» $^$. Так, вместо $X\% * X\%$ мы могли смело написать X^2 . Деление обозначается прямой или обратной наклонной чертой: прямая черта $/$ соответствует обычному делению, а обратная $\backslash$ — делению нацело. Заметим, что Бейсик хорошо приспособлен для математических вычислений: в него встроено ряд математических функций, таких, как тригонометрические функции, логарифмы, экспоненты и др. Например, для вычисления косинуса в программе можно написать $B = \text{COS}(A)$. Как всегда, приоритет операции возведения в степень наибольший, затем идут операции умножения, деления, сложения и вычитания, сравнения и логические операции.

Для всех операций сравнения в Бейсике используется сочетание значков $>$, $<$, $=$. Например, условие « X больше Y » можно записать как $X > Y$, $X >= Y$ означает « X больше.

или равен Y», а условие «X не равен Y» записывается как $X < > Y$.

Кроме того, в Бейсике встречаются логические операции. Они также записываются стандартным образом: AND — логическое И, NOT — отрицание, OR — ИЛИ, XOR — исключающее ИЛИ, EQV — эквивалентность.

Например, для того чтобы записать на Бейсике условие «X больше Y и Y не равно 5», напомним

```
X > Y AND Y < > 5
```

Теперь мы готовы к тому, чтобы перевести на Бейсик программу 5 из гл. 4. Она выглядит следующим образом:

```
10 PRINT
20 INPUT "Введите два целых числа:", A%, B%
30 IF A% > B% THEN MAX% = A% ELSE MAX% = B%
40 PRINT "Максимальное из введенных чисел — "; MAX%
50 END
```

В этой программе нам знакомо все, кроме оператора в строке 30. Это оператор условного перехода. Как видим, в Бейсике он записывается весьма естественным образом. Но, к сожалению, оператор IF-THEN-ELSE в Бейсике ограничен только одной строкой программы.

Рассмотрим другие управляющие структуры Бейсика. Важнейшим оператором Бейсика является безусловный переход GOTO. Этот переход может происходить к заданному номеру строки или к заданной метке (меткой в тексте программы на Бейсике считается текстовое выражение с двоеточием). Например,

GOTO 40 — переход к строке 40

GOTO AAA — переход к метке AAA

Перепишем программу 5, используя оператор безусловного перехода:

```
10 PRINT
20 INPUT "Введите два целых числа: ", A%, B%
30 IF A% > B% THEN GOTO AAA ELSE GOTO 40
40 MAX% = B%
45 GOTO 60
50 AAA: MAX% = A%
60 PRINT "Максимальное из введенных чисел — "; MAX%
70 END
```

Как видим, программа усложнилась. К сожалению, Бейсик устроен так, что без оператора GOTO в нем не обойтись.

Теперь опишем выполнение цикла на Бейсике. Следующий фрагмент программы выполняет операцию сложения элементов массива A десять раз:

```
20 SUM = 0
30 FOR I = 1 TO 10
40 SUM = SUM + A (I)
50 NEXT I
```

Этот фрагмент можно перевести следующим образом: «Сделать операцию 40 для переменной I, изменяющейся от 1 до 10». В цикле FOR-NEXT можно ввести шаг изменения переменной, например

```
FOR I = 1 TO 10 STEP 0.5
```

означает «Сделать от 1 до 10 с шагом 0.5».

Наконец, рассмотрим структуру «Делать до тех пор, пока...». В Бейсике она реализуется с помощью конструкции WHILE-WEND. При этом, если условие цикла истинно, программа выполняет его до тех пор, пока не встретит оператор WEND, затем возвращается к проверке условия и, если оно стало ложным, прекращает выполнение цикла. Например, следующий фрагмент программы будет, как и раньше, суммировать элементы массива A:

```
10 I = 1
20 SUM = 0
30 WHILE I <= 10
40 SUM = SUM + A (I)
50 I = I + 1
60 WEND
```

Теперь мы созрели для того, чтобы записать программу 6 из гл. 4:

```
10 REM Целая переменная X — последнее из введенных
чисел
20 REM Целая переменная MAX — максимальное из введенных чисел
30 REM Строковая пер. ОТВЕТ = «д», если ввод чисел не закончен
40 PRINT
50 INPUT "Введите число"; X%
60 INPUT "Это все (д/н) "; ОТВЕТ$
70 MAX% = X%
80 WHILE ОТВЕТ$ = "н"
90 INPUT "Введите число"; X%
100 INPUT "Это все (д/н) "; ОТВЕТ$
110 IF MAX% < X% THEN MAX% = X%
120 WEND
130 PRINT "Максимальное из введенных чисел = "; MAX%
140 END
```

Мы узнали уже большую часть необходимых сведений о Бейсике. Рассмотрим, каким образом из Бейсика вызвать некоторую функцию. Для этого необходимо написать оператор

GOSUB номер строки

При этом произойдет передача управления к участку программы, начинающемуся с заданной строчки. Этот участок должен располагаться после оператора END. В подпрограм-

мы имеет смысл выделять участки программ, выполняющих какую-либо стандартную функцию. В некоторых компиляторах Бейсика может применяться и другой способ организации подпрограмм с помощью оператора CALL. Оформление фрагментов программы с помощью подпрограмм такого типа наиболее естественно, так как позволяет разбивать большие тексты на отдельные файлы, которые могут раздельно компилироваться, а затем связываться в процессе операции сборки. Однако во многих версиях компиляторов Бейсика при вызове подпрограмм с помощью оператора CALL могут возникнуть разнообразные сложности. Например, в версии 2.0 Бейсика фирмы IBM в подпрограмме почему-то не работает оператор SWAP.

Наконец, рассмотрим работу с функциями. В Бейсике можно определить функцию с помощью команды DEF. Название функции при этом обязательно должно начинаться с сочетания букв FN. Например, функция, вычисляющая гипотенузу треугольника, выглядит на Бейсике следующим образом:

```
DEF FNGIPOT (KATET1, KATET2) = SQR(KATET1^2 + KATET2^2)
```

В этой функции SQR — встроенная в Бейсик функция вычисления квадратного корня. Теперь для вычисления гипотенузы в программе достаточно написать следующую строку:

```
C = FNGIPOT (A, B)
```

Вычисленное значение гипотенузы по катетам, хранящимся в переменных A и B, будет помещено в переменную C. В компиляторе Бейсика можно задать и более сложную функцию, имеющую много строк. При этом определение функции должно заканчиваться сочетанием END DEF. Например,

```
DEF FNGIPOT (KATET1, KATET2)
```

```
K1 = KATET1^2
```

```
K2 = KATET2^2
```

```
GIP = SQR (K1 + K2)
```

```
FNGIPOT = GIP
```

```
END DEF
```

Заметим, что функции в Бейсике нужно определять довольно осторожно: их определение всегда должно предшествовать первому вызову функции. Функция, состоящая из многих строк, может «испортить» переменные, используемые в программе; необходимо следить за соответствием типа функции (целая, строковая, вещественная) и т. д.

Управление ОС и устройствами. Для получения информации о файлах применяется команда FILES «имя файла». В текстовой строке «имя файла» можно пользоваться всеми правилами, принятыми в ДОС, например указывать полный путь по дереву каталогов, использовать звездочки и вопросительный

знак. Так, команда Бейсика FILES "*.BAS" выдаст на экран все файлы текущего каталога с расширением BAS. Для изменения текущего каталога в Бейсике имеется команда CHDIR «имя каталога». Для создания нового каталога достаточно дать команду MKDIR «имя каталога». Для стирания каталога в Бейсике служит команда RMDIR «имя каталога». Для стирания файла служит команда KILL «имя файла».

Сразу заметим, что из Бейсика можно выполнить любую системную команду ДОС с помощью оператора SHELL «текст команды». Например, строка

```
10 SHELL "DIR"
```

выдаст на экран текущий каталог. Если написать команду SHELL без параметров, то мы временно окажемся в операционной системе и сможем давать любые команды ДОС. Для того чтобы вернуться в программу на Бейсике, теперь необходимо написать EXIT. Пользуясь этой командой, мы можем переименовывать файлы, получать информацию об их параметрах, запускать пакетные файлы и т. п.

При разработке программ на Бейсике часто возникает необходимость записывать числовые значения или тексты в файл на диске. Рассмотрим только работу с файлами последовательного доступа. Для этого используются команды

OUTPUT

```
OPEN "имя файла" FOR APPEND AS #номер файла
INPUT
```

Эта команда открывает файл пользователя соответственно для записи, добавления данных и для чтения;

CLOSE #номер файла — закрытие файла,

PRINT #номер файла, список — запись в файл,

WRITE номер файла, список — запись в файл,

INPUT #номер файла, список — чтение из файла.

Структура операторов PRINT и INPUT при этом аналогична их структуре при выводе на экран. Более того, информация в файле будет расположена аналогично тому, как она выглядит на экране. При записи данных в файл более удобно пользоваться оператором WRITE, так как при этом записи не будут путаться при их последующем чтении.

Например, следующий фрагмент программы открывает файл DATA.DAT, записывает в него массив из 100 чисел, закрывает этот файл, снова открывает его для чтения и считывает данные:

```
10 OPEN "DATA.DAT" FOR OUTPUT AS #1
```

```
20 FOR I=1 TO 100
```

```
30 WRITE #1, A(I)
```

```
40 NEXT I
```

```
50 CLOSE #1
```

```
60 OPEN "DATA.DAT" FOR INPUT AS #2
```

```
70 FOR I=1 TO 100
```

```
80 INPUT #2, A(I)
90 NEXT I
100 CLOSE #2
```

В Бейсике используется ряд удобных функций управления экраном и выводом текста. Среди этих функций необходимо прежде всего отметить команду CLS — очистку экрана. Функции задания режима экрана, ширины экрана, выбора страницы экрана в Бейсике устанавливаются при помощи двух команд:

WIDTH 40 или 80

задает ширину экрана;

SCREEN режим, [флаг цвета] [,активная страница] [,визуализируемая страница]

Скобки здесь означают, что соответствующие параметры вводить необязательно. Команда SCREEN позволяет установить следующие режимы:

режим =0 — текстовый,

=1 — цветная графика низкого разрешения 320*
*200 точек,

=2 — черно-белая графика высокого разрешения
640*200;

Флаг цвета =0 — запрещает использование цвета,

=1 — разрешает цвет;

активная страница =0—7 для ширины 40 или 0—3 для ширины 80. В эту страницу памяти будут выполняться все операции вывода на экран;

визуализируемая страница — аналогична активной странице, выбирает страницу, визуализируемую в данный момент.

Например, следующие две команды выбирают цветной текстовый режим 40×25, активную и визуализируемую страницу 0:

```
10 WIDTH 40
```

```
20 SCREEN 0,1,0,0
```

(можно написать просто SCREEN 0,1). Для того чтобы теперь изменить активную и визуализируемые страницы, достаточно дать команду SCREEN „1,0. Заметим, что пропуская параметры в команде Бейсика, мы просто указываем соответствующее количество запятых.

Чтение текущей строки курсора выполняется с помощью функции CSRLIN, например LINE=CSRLIN. Чтение текущего столбца курсора выполняется с помощью функции POS от произвольного аргумента, например COLUMN=POS(0).

Команду позиционирования курсора на экране мы уже рассматривали: LOCATE столбец, строка. С помощью этой команды можно задать и вид курсора. Для этого ее необходимо записать следующим образом:

LOCATE столбец, строка, курсор, начало, конец

В этой команде
курсор = 0 — невидимый,
 = 1 — видимый,
начало — от 0 до 7,
конец — от 0 до 7.

Например, команда LOCATE „,0,7 задаст размер курсора максимально возможным.

Для получения информации о символе, напечатанном на экране, в Бейсике предусмотрена функция SCREEN. Так, если написать A=SCREEN (5, 10), то в переменной A будет содержаться код символа, напечатанного в 10-й позиции 5-й строки экрана.

Для переключения цвета символов в текстовом режиме в Бейсике используется оператор

COLOR цвет символа, цвет фона, цвет рамки,
где цвет — стандартные коды от 0 до 31 (см. гл. 5). Например, дав команду COLOR 2, 0, 4, мы создадим на экране красную рамку и выберем режим «зеленые буквы на черном фоне».

Теперь рассмотрим графические команды Бейсика. Напомним, что для работы в графическом режиме прежде всего необходимо выбрать необходимый режим с помощью команды SCREEN 1 (или 2). Команда «изобразить точку» в Бейсике выглядит как PSET (x, y), цвет. Здесь (x, y) — координаты точки, цвет может изменяться от нуля до 3 в цветном режиме и от 0 до 1 в черно-белом режиме. Для того чтобы получить информацию о цвете точки экрана, используется функция POINT (x, y). Например, команда A=POINT (100, 100) поместит в переменную A значение цвета в точке экрана с координатами (100, 100).

В цветном режиме палитру можно установить с помощью следующей команды:

COLOR цвет фона, палитра
цвет фона — целое число от 0 до 15, палитра — 0 или 1.

В Бейсике предусмотрен прекрасный набор графических процедур, которые мы уже упоминали в § 2.4. Приведем их краткое описание:

LINE (x1, y1) — (x2, y2), цвет, [B[F]]

Скобки здесь означают, что параметры, заключенные в них, можно не указывать. Эта команда может нарисовать линию между точками с координатами (x1, y1) — (x2, y2) заданным цветом. Если включить в нее дополнительную букву B, то будет нарисован прямоугольник, а если написать две буквы BF, то этот прямоугольник будет заполнен выбранным цветом. Примеры применения этой команды:

LINE (10, 10) — (100, 100), 2 — нарисовать красную линию,

LINE (10, 10) — (100, 100), 2, B — нарисовать красный прямоугольник,

LINE (10, 10) — (100, 100), 2, BF — нарисовать закрашенный красный прямоугольник.

Команда

CIRCLE (x, y), r [цвет] [,начало] [,конец] [,отношение] может нарисовать на экране круг, сектор, эллипс. Параметры этой команды имеют следующий смысл: (x, y) — центральная точка окружности; r — радиус; цвет — цвет (как и раньше, 0—3 или 0—1, в зависимости от выбранного графического режима); начало, конец — углы в радианах, между которыми необходимо проводить построение; отношение — отношение радиуса по оси x к радиусу по оси y (применяется при построении эллипсов).

Примеры применения этой команды:

CIRCLE (100, 100), 50, 2 — нарисовать красный круг радиуса 50 с центром в точке (100, 100),

CIRCLE (100, 100), 50, 2...5/18 — нарисовать эллипс с отношением осей x к y как 5/18.

Закраска замкнутых фигур экрана обеспечивается с помощью оператора PAINT (x, y), цвет закрашки, цвет границы. Например, команда PAINT (100, 100), 1, 2 закрасит внутренность нарисованного красного круга зеленым цветом.

Интересную возможность работы с графическими изображениями предоставляют операторы GET и PUT. Они выглядят следующим образом:

GET (x1, y1) — (x2, y2), имя массива

PUT (x1, y1), имя массива, операция

Оператор GET помещает прямоугольную область видеопамати в пределах заданных координат в массив.

Длина массива в байтах при этом $4 + ((\text{размер } x) * (\text{глубина в битах}) + 7) / 8 * (\text{размер } y)$, глубина в битах = 2 для цветного изображения, 1 — для черно-белого.

Оператор PUT помещает изображение из массива в видеопамать, начиная с верхнего левого угла прямоугольной области. При этом можно выполнить ряд логических функций между тем изображением, которое уже имеется на экране, и тем, которое хранится в массиве. Например,

PUT (100, 100), ARRAY, PSET — накладывает изображение, стирая старое,

PUT (100, 100), ARRAY, OR — накладывает новое изображение «поверх» старого.

Можно использовать также коды операций PRESET (создает дополнительное изображение), XOR (исключающее или), AND (логическое И).

Удобные возможности по преобразованию координат экрана предоставляет специальная команда Бейсика WINDOW. Эта команда записывается следующим образом:

WINDOW (x1, y1) — (x2, y2),

где $(x1, y1)$ и $(x2, y2)$ — действительные числа, определяющие новые координаты крайних точек экрана. Так, команда

WINDOW (0, 0) — (320, 200)

сохранит координаты экрана неизменными. Но если дать команду WINDOW (0, 0) — (10, 10), то, построив исходное графическое изображение, мы увидим увеличенную часть верхнего угла экрана размером 10×10 точек. Естественно, что при этом все фигуры, проходящие через этот участок экрана, окажутся растянутыми на весь экран.

Кроме этих команд, в Бейсике имеется специальный язык описания графических объектов, реализуемый с помощью оператора DRAW. Например, команда

DRAW "R50U50"

нарисует на экране две прямых линии: на 50 точек вправо от текущей (R50) и еще на 50 точек вверх от последней точки (U50). В операторе DRAW используется широкий набор команд для проведения линий, которые мы рассматривать здесь не будем.

Читателя, заинтересовавшегося примером простой программы, выполняющей графические построения, мы отсылаем к § 2.4, в котором был приведен фрагмент программы, выполняющей построение графика.

Мы уже рассмотрели команды ввода с клавиатуры INPUT, INPUT\$, INKEY\$, LINE INPUT. Кроме этих функций, в Бейсике также предусмотрены специальные команды для работы с функциональными клавишами F1—F10 и с регистрами клавиатуры. Все эти команды начинаются с сочетания KEY. Рассмотрим их. Команда

KEY номер, текстовая строка
присваивает соответствующей функциональной клавише (с 1 по 10) текстовое выражение. Например,

KEY 2, "YES" — при нажатии на функциональную клавишу F2 мы имитируем ввод с клавиатуры сочетания YES. Эту команду можно немного усложнить, например,

KEY 2, "YES" + CHR\$(13) — эквивалентную вводу с клавиатуры YES и возврату каретки (код 13).

Кроме десяти функциональных клавиш, в Бейсике можно определить дополнительно шесть клавиш с номерами от 15 до 20. Для этого применяется команда

KEY номер, CHR\$(флаг клавиатуры) + CHR\$(скэн-код)
В этой команде флаг клавиатуры — это специальный код, определяющий состояние регистров клавиатуры (нажат или нет верхний регистр, клавиши Ctrl, Alt, NumLock). Скэн-код — просто код символа на клавиатуре. Эти коды мы приводить не будем, запишем только команду, с помощью которой клавише 17 присваивается значение одновременно нажатых клавиш Ctrl и P:

KEY 17, CHR\$(&H04 + &H03) + CHR\$(25)

Здесь сочетание &НО4 (нажата клавиша Ctrl), &НО3 (нажат верхний регистр) — шестнадцатеричные числа, приводимые в описании Бейсика. Код 25 — скэн-код клавиши «Р».

В Бейсике для клавиш, соответствующих перемещению курсора, приняты стандартные номера: 11 — вверх, 12 — влево, 13 — вправо, 14 — вниз.

Мы научились определять двадцать различных клавиш. Теперь их можно использовать в программе. Для этого в Бейсике принято специальное соглашение: вначале нужно задать клавиши, затем написать оператор ON KEY (n) GO SUB, а в следующей строке «включить» соответствующую клавишу командой KEY ON. Например, задав клавишу 17, мы можем написать

```
ON KEY (17) GOSUB 1000  
KEY (17) ON
```

В программе при нажатии клавиши Ctrl-P будет происходить переход к подпрограмме 1000. Если мы хотим произвести какое-то действие с помощью нажатия на одну из десяти функциональных клавиш, которым не было сопоставлено никаких символьных выражений, можно просто написать

```
ON KEY (3) GOSUB 3000  
KEY (3) ON
```

Заметим, что при использовании компилятора компиляцию программы придется проводить со специальным флажком /V или /W. Любую из используемых клавиш можно выключить, дав команду KEY (номер) OFF

Для вывода данных на принтер в Бейсике используется несколько простых команд. Команда

LPRINT текстовая строка

печатает данные на принтере точно так же, как на экране в режиме 80 символов в строке. В текстовую строку можно добавить стандартные для большинства принтеров управляющие символы: возврат каретки CHR\$(13), прогон страницы CHR\$(12) и др. Например, если принтер включен и готов к приему данных, команда

LPRINT «Печатаем на принтере»; CHR\$(12)

напечатает фразу и прогонит страницу бумаги. Другой способ вывода данных на принтер — открыть файл со стандартным именем LPT1: и выводить данные в этот файл. Например,

```
20 OPEN "LPT1:" FOR OUTPUT AS #1  
30 PRINT #1, "ЭТОТ ТЕКСТ ВЫВЕДЕН НА ПРИНТЕР"  
40 CLOSE #1
```

Управление выводом данных осуществляется с помощью посылки в принтер стандартных кодов, описанных в руководстве по использованию конкретного печатающего устройства, или с помощью команды DOC MODE (см. § 3.1). В Бейсике преду-

смотрена также специальная функция для считывания положения печатающей головки принтера LPOS. Например, строка программы.

A=LPOS (0)

поместит в переменную A текущую позицию печатающей головки принтера.

В Бейсике предусмотрены две простые функции для управления таймером. Первая из них читает или устанавливает время. Эта функция выглядит следующим образом:

A\$=TIMES\$ — чтение времени в строковую переменную A\$

TIMES\$=A\$ — установка времени из строковой переменной

A\$

Переменная A\$ выглядит следующим образом:

часы:минуты:секунды

Например, программа

10 A\$=«12:10:00»

20 TIMES\$=A\$

устанавливает текущее время равным 12 ч 10 мин.

Вторая функция Бейсика читает время в секундах, прошедшее с момента запуска таймера. Вот как она выглядит в типичном фрагменте программы:

30 TIMER ON (запуск таймера)

150 A=TIMER (чтение времени)

400 TIMER OFF (выключение таймера)

Для управления синтезатором в Бейсике можно с успехом ограничиться командой

SOUND частота Гц, длительность,

где частота лежит в диапазоне от 37 до 32767 Гц, длительность от 0,015 до 65535. Длительность=1 соответствует примерно 1/18 с. Кроме этого, в Бейсике имеются команды BEEP (сигнал 800 Гц четверть секунды) и PLAY. Команда PLAY позволяет записывать мелодии, используя условные обозначения нот, и рассматривать ее мы не будем.

Перейдем к командам работы с портами и памятью. В Бейсике используются следующие команды:

DEF SEG=сегмент

задает сегментную часть адреса. Для задания адреса можно использовать шестнадцатеричное число, например, DEF SEG==&H449;

содержимое ячейки памяти считывается с помощью функции PEEK (смещение);

например, A=PEEK (156);

записать в память байт по указанному адресу можно с помощью команды

POKE смещение, число

Для чтения байта из порта служит функция

A=INP (порт)

Для записи в порт служит функция

OUT порт, число

Хотя мы рассмотрели большую часть команд Бейсика, часть из них попала в наше описание. Кроме того, в Бейсике имеется множество удобных функций (математических, строковых и др.), знакомство с которыми также необходимо. Поэтому читатель, решивший писать программы на Бейсике, рано или поздно должен прштудировать более подробные описания конкретной версии Бейсика. Мы советуем также внимательно прочесть описание имеющихся у читателя версий компилятора и интерпретатора Бейсика: не исключено, что в них внесены модификации и изменения.

В заключение выскажем несколько субъективных соображений по поводу программирования на Бейсике, неочевидных при том поверхностном рассмотрении этого языка, которое мы провели. Бейсик является одним из самых контрастных и противоречивых языков программирования для ПЭВМ. Приведем несколько таких свойств Бейсика:

изучить Бейсик начинающему пользователю пожалуй проще, чем любой другой язык, но работать на нем в итоге оказывается в несколько раз сложнее, чем на других языках высокого уровня;

в Бейсике соседствуют чрезвычайно простые, удобные и отточенные инструкции языка с крайне сложными, запутанными и неестественными командами;

Бейсик сочетает прекрасные средства для ведения диалога и графики, управления устройствами ПК с непродуманными и нелогичными управляющими структурами.

Интересно, что из-за этих свойств программисту, знающему любой язык высокого уровня, изучить Бейсик как правило сложнее, чем любой другой язык.

Вначале рассмотрим те команды Бейсика, которые, по нашему мнению, удобны. Несомненным удобством и простотой отличаются операторы ввода-вывода, прежде всего операторы PRINT, INPUT и LOCATE. Эти операторы, как мы убедились, позволяют без особых хлопот организовывать диалог с пользователем, размещать текст в любом месте экрана. Удобны и просты также функции работы с текстовыми переменными. Например, конкатенация текста в Бейсике выполняется просто «сложением» двух текстовых переменных. Пусть переменная A1\$ = «Конка», A2\$ = «тенация». Тогда операция A3\$ = A1\$ + A2\$ поместит в переменную A3\$ текст «Конкатенация». Достаточно разумно в Бейсике сделаны операции с циклами (конструкция FOR—NEXT и WHILE—WEND). В общем-то просто устроена запись в файлы с последовательным доступом. Безусловным достоинством Бейсика является наличие большого числа функций для отлавливания программой состояния функциональных клавиш (команды ON KEY), ошибок и для управления громкоговорителем (SOUND). Но самым большим пре-

имуществом Бейсика, резко выделяющим его из других языков программирования для ПЭВМ, является набор графических команд, непосредственно встроенных в язык и предоставляющих пользователю широкие возможности программирования графических функций. Этот набор команд, по нашему мнению, представляет собой «золотую середину» между сложными графическими пакетами программ и набором простейших графических примитивов. Графические команды Бейсика крайне просты, записываются довольно естественным образом и содержат все необходимое для создания графических изображений хорошего качества. Как нам кажется, именно набор графических команд привел к столь широкому распространению этого языка на ПЭВМ. В целом Бейсик является единственным языком программирования для ПЭВМ, в который так полно включены функции взаимодействия с ДОС и периферией компьютера.

Все контрасты Бейсика, по-видимому, связаны с тем, что он создавался довольно хаотичным образом в отличие от таких языков, как Паскаль и Си. Появившись на ранних этапах развития ЭВМ как язык для обучения программированию, Бейсик обладал действительно значительным удобством ввода и вывода данных, работы с текстовыми переменными. Благодаря этому он вскоре приобрел значительную популярность и стал «обрастать» усовершенствованиями, которые, однако, проводились в стиле строительства Вавилонской башни и напоминали заплатки на рваной простыне. Это отчасти было вызвано желанием разработчиков сохранить совместимость новых и старых версий Бейсика и в итоге привело к тому, что язык пестрит нелогичностями, которые вызывают по меньшей мере удивление у квалифицированного программиста. Ни в одном другом языке не существует столько подводных камней для разработчика программы. Рассмотрим несколько наиболее типичных нелепостей, которые подстерегают программиста, решившего писать программы на Бейсике.

Возьмем, например, странное соглашение о том, что по умолчанию все переменные языка являются действительными числами. Даже счетчики циклов являются действительными, хотя трудно предположить, что кому-нибудь понадобится выполнить операцию сложения 7,3 раза. Это открывает огромный простор для всякого рода ошибок и погрешностей, не говоря уже о том, что для объявления целых чисел всюду в тексте программы приходится писать странные сочетания типа `I%` вместо `I`. Теперь достаточно в одном месте забыть поставить знак процента, и программа не будет работать. В большинство версий Бейсика включен специальный оператор объявления целого типа чисел `DEFINT`, который может описать все переменные, начинающиеся с заданных букв, как целые. Но такая двойная возможность объявления типов переменных приводит только к путанице в программе.

Другая странность языка — операция деления. В Бейсике имеется две операции деления: просто деление (символ «/») и деление нацело (символ «\»), которые могут приводить к совершенно различным результатам. Простейший пример такого рода. Пусть переменная N описана как целое число и $N=33$. Тогда выражение $N=N\backslash 2$ присвоит N значение 16, а выражение $N=N/2$ присвоит N значение 17. Этот результат совершенно непонятен, так как в обоих случаях вычисления производятся с целым числом, и он может привести к серьезным ошибкам в программах, в чем мы убедились на своем печальном опыте. Наконец, крайне неудобна, на наш взгляд, практика присваивания номеров строкам: это резко ограничивает возможности редактирования программы, так как порядок номеров приходится изменять. В последних версиях компиляторов Бейсика можно не нумеровать строки, но при этом иногда возникают противоречия, связанные с тем, что многие операторы рассчитаны на обязательную нумерацию строк.

Одним из самых больших недостатков Бейсика является непродуманная конструкция оператора условного перехода. В Бейсике имеется вроде бы все необходимое для выполнения этой операции (оператор IF-THEN-ELSE), но этот оператор почему-то ограничен одной строкой программы! Это означает, что мы можем записать только очень короткие условия, состоящие максимум из трех-четырех небольших действий, например

```
IF X=1 THEN X=X+1:A=A-1 ELSE X=0
```

Практика показывает, что условный переход играет одну из самых решающих ролей в конструировании программы и такая «обрубленная» форма этого оператора в Бейсике заставляет применять множество операторов GOTO, что сразу же запутывает программу. Понятно, что сколько-нибудь большую программу на Бейсике написать очень трудно.

Значительные неудобства могут возникнуть и при разработке программы, состоящей из нескольких подпрограмм. В Бейсике до недавнего времени использовался только один способ вызова подпрограммы — при помощи операторов GOSUB и RETURN. При таком способе все подпрограммы должны были находиться в одном и том же файле и, более того, все они использовали те же переменные, что и главная программа, т. е. не имели формальных параметров. При такой организации программы возникает серьезная опасность того, что подпрограмма может испортить значения переменных, необходимых в главной программе. Те же соображения относятся и к функциям Бейсика, не говоря уже о том, что имена функций обязательно должны начинаться с сочетания FN.

Кроме того, при таком способе организации программ возникает необходимость по мере работы главной программы передавать управление другим программам с помощью спе-

циального оператора CHAIN. Это сразу же приводит к появлению ряда дополнительных трудностей, связанных с передачей общих переменных из одной программы в другую. В последних версиях компиляторов Бейсика появилась возможность организовывать «настоящие» подпрограммы, имеющие формальные параметры, которые могут храниться в отдельных файлах. Эти подпрограммы вызываются с помощью оператора CALL. Но и здесь возникает ряд ограничений. Например, если в подпрограмму передается целая переменная, то в описании подпрограммы она обязательно должна сопровождаться значком «%». Кроме этого, почему-то нельзя указывать одну и ту же переменную в списке формальных параметров при вызове подпрограммы. Это означает, что если у нас имеется подпрограмма AAA (X1, X2, X3), то мы не имеем права написать CALL AAA(I, I, J), иначе работа программы станет непредсказуемой. Одним словом, сон разума рождает чудовищ.

И уж совсем странно в Бейсике организована работа с файлами с произвольным доступом. Вместо ясной и понятной процедуры, которая позволяет открыть файл и записать в него заданное количество чисел, указав при этом номер записи в Бейсике ПЭВМ этот процесс разбит на 4 ступени:

- 1) открыть файл (оператор OPEN),
- 2) объявить структуру записи (оператор FIELD),
- 3) присвоить значения переменным, указанным в операторе FIELD (операторы LSET и RSET),
- 4) записать или считать данные (операторы PUT и GET).

Кроме того, если в файл записываются числа, то при этом оказывается необходимо преобразовать числовые значения в строковые переменные.

Кроме этих трудностей, которые представляются нам самыми принципиальными, существует еще множество странных свойств языка, которые читатель без труда обнаружит в процессе отладки своих собственных программ. Но, несмотря на них, если бы появилась версия Бейсика для ПЭВМ, в которой хотя бы была улучшена структура оператора IF-THEN-ELSE и продумана возможность структурирования программы, мы бы безусловно рекомендовали Бейсик всем начинающим (и не только начинающим) программистам ПК.

Самое неприятное при работе с Бейсиком состоит в том, что пользователь, научившийся работать на нем, свято убежден, что Бейсик в действительности очень простой и хороший язык, и те трудности, которые он иногда испытывает, только возрастут при переходе к другим языкам программирования. Кроме того, Бейсик настолько разительно отличается от структурных языков программирования, что «преодолеть» барьер изучения другого языка после Бейсика довольно тяжело, особенно если не осознать, насколько это может облегчить жизнь. Но объективная реальность такова, что для сколько-нибудь

серьезного программирования пользователю рано или поздно придется отказываться от Бейсика, теряя при этом все его преимущества, но приобретая взамен силу структурных языков программирования. Мы надеемся, что наша книга поможет в этом читателю.

6.4. НЕСКОЛЬКО ЗАМЕЧАНИЙ ПО ПОВОДУ ЯЗЫКА ФОРТРАН

Фортран является одним из самых неудобных языков для работы на ПЭВМ, но чаще многих других языков используется для решения вычислительных задач. Неудобства работы на Фортране вызваны:

разнообразием версий Фортрана ПЭВМ, плохо совместимых друг с другом и со стандартом Фортрана, обособленностью Фортрана от периферийных устройств ПК: в большинстве версий этого языка отсутствуют функции работы с экраном, графическими возможностями, таймером, памятью, взаимодействие с ДОС и т. п.

Широкое использование Фортрана для математических и инженерных расчетов вызвано прежде всего историческими традициями: существуют огромные пакеты для научных расчетов и численного анализа, написанные на Фортране. Естественно, что инженер, особенно привыкший к Фортрану на других компьютерах и обнаруживающий на ПЭВМ знакомые ему библиотеки программ численного анализа, крайне неохотно расстанется с этим языком.

Первоначально мы планировали совсем не касаться работы с Фортраном на ПК. Но в связи с изложенным приняли компромиссное решение: не описывать подробно сам язык (в литературе более чем достаточно книг по программированию на Фортране, и мы предполагаем, что пользователь Фортрана ПЭВМ уже знает этот язык), но кратко рассмотреть особенности работы с ним на ПЭВМ.

Вначале коснемся разнообразия версий Фортрана. Для компьютеров, совместимых с IBM PC XT, существует множество различных компиляторов Фортрана: IBM Fortran, Microsoft Fortran, Professional Fortran, Supersoft Fortran, стандартный Фортран-77 и др. Рассмотрим для примера работу с компилятором Фортрана фирмы IBM, версия 2. Этот компилятор записан на трех дисках. На первом из них содержатся файлы

FOR1.EXE — компилятор первого прохода, находит синтаксические ошибки в программах,

SPSETUP.BAT — программа автоматической загрузки Фортрана в компьютер и ряд служебных файлов.

Второй диск содержит следующие файлы:

FOR2.EXE — компилятор второго прохода, создает объектный файл,

FOR3.EXE — компилятор третьего прохода, может использоваться для получения листинга объектного файла,

LINK.EXE — линкер,

На третьем диске записаны библиотеки Фортрана:

8087ONLY.LIB — используется, если в ПЭВМ установлен математический сопроцессор,

EMULATOR.LIB — библиотека для работы без сопроцессора, может использоваться и при его наличии.

На этом диске также записан ряд служебных файлов для создания основной фортрановской библиотеки, которая называется FORTRAN.LIB.

Описываемый компилятор Фортрана устроен таким образом, что при его установке в компьютер необходимо обязательно пройти стандартную последовательность операций, записанных в файле SPSETUP.BAT. Только в результате работы этой программы на фиксированном диске или на дискете будут созданы необходимая файловая структура и библиотеки для работы с Фортраном. Что же нужно сделать для того, чтобы откомпилировать программу? Вот типичная последовательность операций:

FOR1 TEST; (первый проход компилятора)

FOR2 TEST; (второй проход компилятора)

На этом шаге компилятором будет создан объектный файл. Теперь мы можем вызвать линкер:

LINK TEST;

В результате работы линкера будет создан выполнимый файл, который можно запускать, набрав его имя на клавиатуре и дав возврат каретки.

По-другому устроена компиляция в системе Microsoft Fortran, версия 3 и 4. Поэтому при запуске Фортрановских программ на ПЭВМ необходимо подробно ознакомиться с конкретной версией компилятора этого языка.

Теперь поговорим о языковых различиях Фортранов. В описанных нами версиях компиляторов существует ряд существенных отличий от стандарта Фортрана-77. Например, в Фортране IBM нельзя использовать:

комплексный тип переменных (COMPLEX),

оператор BLOCK DATA,

операции конкатенации символьных переменных,

оператор ENTRY,

операторы READ и PRINT,

логические операторы EQV и NEQV,

оператор DATA в тексте программы после задания спецификаций массивов и переменных.

Кроме того, в данной версии Фортрана размер массивов данных почему-то не должен превышать 64 Кбайт, но допускается использование нескольких поименованных COMMON-блоков, каждый из которых может иметь размер до 64 Кбайт.

Единственной «системной» процедурой Фортрана IBM являются две подпрограммы для считывания текущего времени и даты. Большим достоинством этого Фортрана, однако, является то, что в комплекте с ним поставляется библиотека более чем 150 научных подпрограмм.

Фортран фирмы Microsoft, версия 3, является более продвинутым, чем только что описанная версия Фортрана IBM. Эта версия Фортрана полностью совместима с Паскалем той же фирмы: компилятор первого прохода Фортрана и Паскаля создают промежуточный текст на одном и том же языке, так что второй и третий проходы компиляторов Фортрана и Паскаля фирмы Microsoft идентичны. Кроме того, в Фортран Microsoft третьей версии включена возможность вызова некоторых процедур языка Си, например временная передача управления операционной системе или программе на языке Си.

Фортран фирмы Supersoft, как нам показалось, является довольно хорошо «приспособленным» к работе с ПЭВМ. Его основные преимущества:

- соблюден стандарт Фортрана-77,
- можно использовать всю оперативную память ПЭВМ,
- включены функции для интерфейса программ на Фортране к программам на языках Си и Бейсик,
- в Фортран встроено несколько процедур для работы с памятью и портами.

Самым развитым Фортраном для ПЭВМ является пакет фирмы Microsoft, версия 4. О нем хотелось бы сказать особо. Если читатель сможет «выбирать» фортрановскую систему для своей ПЭВМ, то, конечно, лучшей версии пока нам неизвестно. Этот Фортран занимает 7 дискет и содержит диалоговый отладчик, аналогичный отладчику той же фирмы для языка Си. На семи дискетах содержится огромное число файлов: автоматическая установка Фортрана на ПЭВМ, компиляторы первого, второго и третьего проходов, линкер, библиотеки для различных моделей памяти, примеры программ, служебные программы для сжатия файлов, содержащих коды, всего больше 130 файлов! Честно говоря, огромные размеры этого Фортрана подавляют. Для описания даже малой части функций управления компилятором необходимо было бы потратить значительный объем книги и мы предоставляем это сделать заинтересовавшемуся пользователю. У Фортрана этой версии имеется два огромных преимущества перед остальными:

- полная поддержка всех свойств Фортрана-77,
- полная совместимость со всеми библиотеками языка Си фирмы Microsoft, версия 4.

Это означает, что из Фортрана можно вызвать любую функцию или подпрограмму языка Си. Для этого в данной версии Фортрана предусмотрены специальная команда INTER-FACE, которая описывает в фортрановской программе ссылку

на подпрограммы, и функции библиотек Си. Заметим, что для того чтобы правильно описать эти ссылки, необходимо выучить целый ряд правил, устанавливающих соответствие типов переменных Си и Фортрана. Эти правила не всегда просты и процесс вызова программ Си из Фортрана довольно громоздок. Но как бы то ни было, в настоящее время это единственная возможность вызвать из Фортрана системные функции, описанные в разделе, посвященном языку Си. Кстати, этот Фортран чаще всего используется «наоборот»: для вызова фортрановских научных подпрограмм из управляющей программы, написанной на Си.

Таким образом, большинство версий Фортрана позволяют проводить на ПЭВМ математические расчеты и простейший диалог. Пользователь, привыкший решать счетные задачи на Фортране, без особого труда приспособится к работе с ним на ПЭВМ. Программист, решивший написать более серьезную прикладную программу, которая хотя бы строит график на экране, начнет сталкиваться с трудностями. В настоящее время нам известно три графических пакета, которые могут быть использованы из Фортрана ПЭВМ: пакет Halo (см. § 2.4), система GKS (Graphics Kernel System) и пакет «Графор». Но для нас остается совершенно непонятным, почему в Фортран ПЭВМ не включают десять—пятнадцать стандартных подпрограмм для управления экраном и графикой — это бы значительно расширило его возможности и не заставило бы пользоваться монстрами типа четвертой версии Фортрана Microsoft.

Трудно сказать, какая из описанных версий Фортрана получила наибольшее распространение на ПЭВМ. Безусловно, Фортран будет широко использоваться на ПЭВМ для научных расчетов. Быть может, в будущем появится версия этого языка, более удобная для работы, чем существующие в настоящее время.

Как видим, пользователь, переносящий свои программы на ПЭВМ, должен быть очень аккуратен: нет никакой гарантии того, что его фортрановские программы могут без изменений заработать на ПЭВМ. В такую же ситуацию попали и авторы книги: при переносе ряда стандартных программ для обработки сигналов на ПК нас больше всего расстроило отсутствие в Фортране IBM комплексного типа переменных. Из-за этого пришлось переписывать большие фрагменты программ. Единственный совет, который можно дать пользователю, использующему Фортран, — внимательно прочтите описания вашей версии Фортрана, особенно раздел «Отличия от стандарта Фортрана-77» и только потом начинайте перенос текстов программ на ПК.

ПРИЛОЖЕНИЕ

ПРОГРАММЫ НА ЯЗЫКЕ СИ

```
/* файл USER.H */

/* Описания процедур, которых нет */
/* в стандартной библиотеке */

/* Файловая система */
extern int file_attr( char[]);
extern long size_of_file ( char[]);
extern long time_of_file ( char[]);
extern set_file_attr( char[], int, int, int *);
extern start_find ( char[], int, int *);
extern int is_next_file();
extern next_file(char[]);

/* Клавиатура */
extern unsigned int a86(int,int,int,int,int);
/* intnum ah al cx dx */
#define k_reg() a86(22, 2, 0, 0, 0)
#define k_ascii() (a86(22, 1, 0, 0, 0)&255)
#define k_extc() (a86(22, 1, 0, 0, 0)>>8)
#define k_pop() a86(22, 0, 0, 0, 0)
#define k_flush() bdos(12,0,0)

/* Экран */
static struct vpg { struct { char s,m;} c[25][80];};

static struct vpg *vmptr =(struct vpg *)0x0B00000001;
static struct vpg *vOptr =(struct vpg *)0x0B80000001;
static struct vpg *v1ptr =(struct vpg *)0x0B90000001;
```



```

static struct vpg *v2ptr =(struct vpg *)0x0BA0000001;
static struct vpg *v3ptr =(struct vpg *)0x0BV00000001;

#define VM vmptr->c /* монохромный адаптер */
#define VM0 v0ptr->c /* ЦГА, страница 0 */
#define VM1 v1ptr->c /* ЦГА, страница 1 */
#define VM2 v2ptr->c /* ЦГА, страница 2 */
#define VM3 v3ptr->c /* ЦГА, страница 3 */

extern unsigned int v86dx(int,int,int);
/* ah, dh, dl */
extern unsigned int v86bh(int,int,int);
/* ah, bh, bl */

#define s_mode() (a86(16,15, 0, 0, 0)&255)
#define s_setmode(m) a86(16, 0, (m), 0, 0)
#define s_page() v86bh(15, 0, 0)
#define s_setpage(m) a86(16, 5, (m), 0, 0)
#define s_poscur(l,c) v86dx(2, 1, c)
#define s_curline() (v86dx(3, 0, 0)>>8)
#define s_curcol() (v86dx(3, 0, 0)&255)
#define s_drawdot(l,c,clr) a86(16,12, (clr), (c), (l) )
#define s_dot(l,c) (a86(16,13, 0, (c), (l))&255)
#define s_setback(n) v86bh(11, 0, (n))
#define s_setpal(n) v86bh(11, 1, (n))
#define s_off() {while(! (inp(0x3da)&8)){};outp(0x3d8, 0x25);}
#define s_on() {outp(0x3d8, 0x29);}

/* Принтер */
#define lpt_begin(n) a86(23, 1, 0, 0, (n))
#define lpt_send(n,c) (a86(23, 0, (c), 0,(n))>>8)
#define lpt_status(n) (a86(23, 2, 0, 0,(n))>>8)

/* Звук */

```

```

#define oscill_on()    outp(97; inp(97)|1)
#define oscill_off()  outp(97, inp(97)&254)
#define speaker_on()  outp(97, inp(97)|2)
#define speaker_off() outp(97, inp(97)&253)
#define set_period(i) {outp(67, 182);outp(66,(i)&255); \
                        outp(66,(i)>>8);}

/*
    Оставшаяся часть текста нужна
    только при работе с компилятором Microsoft
*/

/* Файловая система */
#define FA_DIREC  0x10 /* Каталог      */
#define FA_RDONLY 0x01 /* Защищен от изменения */
#define FA_HIDDEN 0x02 /* Скрытый      */
#define FA_ARCH   0x20 /* Архивный     */
#define FA_SYSTEM 0x04 /* Системный   */
#define FA_LABEL   0x08 /* Метка тома  */

#define setdisk(d) bdos(14, (d), 0)
#define getdisk()  bdos(25, 0, 0)

/* Память и порты */
#define MK_FP(seg,ofs) ((void far *) \
                        (((unsigned long)(seg) << 16) | (ofs)))

#define poke(a,b,c)    (*((int far*)MK_FP((a),(b))) = (int)(c))
#define pokeb(a,b,c)   (*((char far*)MK_FP((a),(b))) = (char)(c))
#define peek(a,b)       (*((int far*)MK_FP((a),(b))))
#define peekb(a,b)      (*((char far*)MK_FP((a),(b))))

```

```
/* файл LOW.C */
```

```
/*
```

описание интерфейса с BIOS, используемого в макроопределениях команд работы с устройствами в USER.H. Этот файл следует откомпилировать, после чего выполняемый файл из Вашей программы можно получать так:

```
LINK MYPROG+LOW;
```

```
*/
```

```
#include "dos.h"
```

```
unsigned int a86( i, ah, al, cx, dx)
```

```
int i, ah, al, cx, dx;
```

```
{
```

```
    union REGS inr, outr;
```

```
    inr.h.ah = ah; inr.h.al = al;
```

```
    inr.x.bx = 0;
```

```
    inr.x.cx = cx; inr.x.dx = dx;
```

```
    int86 ( i, &inr, &outr);
```

```
    return (outr.x.ax);
```

```
}
```

```
unsigned int v86dx( ah, dh, dl)
```

```
int ah, dh, dl;
```

```
{
```

```
    union REGS inr, outr;
```

```
    inr.h.ah = 15; int86 (16, &inr, &outr);
```

```
    inr.h.ah = ah; inr.h.bh = outr.h.bh;
```

```
    inr.h.dh = dh; inr.h.dl = dl;
```

```

        int86 (16, &inr, &outr);

        return (outr.x.dx);
    }

    unsigned int v86bh(ah, bh, bl)
    int ah, bh, bl;
{
    union REGS inr, outr;

    inr.h.ah = ah;

    inr.h.bh = bh; inr.h.bl = bl;

    int86 (16, &inr, &outr);

    return (outr.h.bh);
}

```

C:\> dial

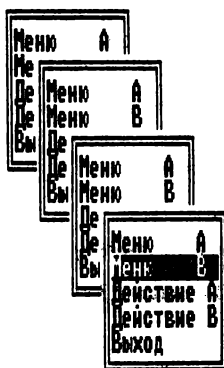


Рис. III. Экран ПЭВМ при выполнении программы диалога

```
/* файл FILE.C */
```

```
/*
```

описания подпрограмм работы с файлами, которые описаны в 6.2, но не содержатся в стандартной библиотеке. Для того, чтобы пользоваться этими подпрограммами, следует откомпилировать этот файл. Если Ваша программа называется, например MYPROG, то выполняемый файл из нее надо получать так:

```
MSC/AL MYPROG;  
LINK MYPROG+FILE;
```

```
*/
```

```
#include "dos.h"  
#include "types.h"  
#include "stat.h"  
#include "stdlib.h"
```

```
file_attr( s ) unsigned char s[];
```

```
{  
    struct WORDREGS inr, outr;  
    struct SREGS seg;  
    inr.ax = 0x4300;  
    inr.dx = FP_OFF ( s );  
    seg.ds = FP_SEG ( s );  
    intdosx( &inr, &outr, &seg);  
    if ( outr.cflag ){ puts("file_attr: ошибка"); exit (1);}  
    return ( outr.cx );  
}
```

```

long size_of_file ( s ) unsigned char s[];
{
    struct stat b;
    if ( stat( s, &b)){ puts("size_of_file: ошибка"); exit(1);}
    return ( b.st_size );
}

long time_of_file ( s ) unsigned char s[];
{
    struct stat b;
    if ( stat( s, &b)){ puts("time_of_file: ошибка"); exit(1);}
    return ( b.st_mtime );
}

set_file_attr( s , q, answer, done )
unsigned char s[]; int q, answer, *done;
{
    struct WORDREGS inr, outr;
    struct SREGS seg;
    inr.ax = 0x4300;
    inr.dx = FP_OFF ( s );
    seg.ds = FP_SEG ( s );
    intdosx( &inr, &outr, &seg);
    if ( answer ){
        inr.cx = (outr.cx | q);
    }else{
        inr.cx = (outr.cx & ~q);
    }
    inr.ax = 0x4301;
    intdosx( &inr, &outr, &seg);
    *done = !outr.cflag;
}

```

```
/* файл FIND.C */
```

```
/*
```

Описание подпрограмм поиска файлов в заданном каталоге. Правила работы с этими подпрограммами объяснены в гл. 5.1 и 6.2. Для того, чтобы пользоваться этими подпрограммами, следует откомпилировать этот файл. Если Ваша программа называется, например MYPROG, то выполняемый файл из нее надо получать так:

```
MSC/AL MYPROG;  
LINK MYPROG+FIND;
```

```
*/
```

```
#include <dos.h>  
#include <errno.h>  
#include <stdlib.h>  
#include <stddef.h>  
static unsigned char buf[43];  
static unsigned int next=0;  
  
static find ( code, s , attr, flag, ax)  
unsigned char s[]; int attr, code, *flag, *ax;  
{  
    struct WORDREGS r;  
    struct SREGS seg;  
    unsigned int dtasegm, dtaoffs;  
    r.ax = 0x2F00;  
    intdosx( &r, &r, &seg);
```

```

    segread ( &seg );
    dtasegm = seg.es; dtaoffs = r.bx;
    /* (dtasegm, dtaoffs) = current DTA */
    r.ax = 0x1A00;
    r.dx = FP_OFF (&buf[0]);
    seg.ds = FP_SEG (&buf[0]);
    intdosx( &r, &r, &seg);
    /* current.DTA = buf */
    r.ax = code;
    r.dx = FP_OFF ( s );
    seg.ds = FP_SEG ( s );
    r.cx = attr;
    *ax = intdosx( &r, &r, &seg);
    *flag = r.cflag;
    /* file found and stored in buf */
    r.ax = 0x2F00;
    r.dx = dtaoffs;
    seg.ds = dtasegm;
    intdosx( &r, &r, &seg);
    /* DTA restored */
}

start_find ( 's' , attr, done)
unsigned char s[]; int attr, *done;
{
    int flag, ax;
    find ( 0x4E00, s, attr, &flag, &ax);
    next = !flag;
    *done = next || (ax == ENMFILE);
}

is_next_file(){ return ( next );}

```



```

next_file ( s )
unsigned char s[];
{
    int flag, ax;
    if (!next){ puts("next_file: ошибка");exit(1);}
    memcpy( s, &buf[30], 13);
    find ( 0x4F00, NULL, 0, &flag, &ax);
    if (flag && ax != ENMFILE){
        puts("next_file: внутренняя ошибка"); exit(1);
    }
    next = !flag;
}

```

```

/* Файл DIAL.C */

/*-----*/
/* Эта программа моделирует диалог с использованием меню. */
/* При этом поддерживаются следующие, обычные для диалоговых */
/* программ, услуги: */
/*      1. Эффект "наложения" окна с меню на текущее изобра- */
/* жение. При выходе из соответствующего меню окно с его */
/* изображением стирается, и в этом месте восстанавливается */
/* то, что там было до входа в меню. */
/*      2. Возможность выбора варианта путем нажатия верти- */
/* кальных стрелок с подсветкой текущего варианта выделен- */
/* ным цветом. Завершается выбор нажатием возврата каретки. */
/*      3. Возможность создания вложенных меню. */
/*      4. Возможность поместить текущее меню в любое место */
/* экрана, включив режим ScrollLock и нажимая на стрелки. */
/* Программа использует библиотеку работы с устройства- */
/* ми, находящуюся в файле LOW.C. То есть выполняемый файл */
/* должен создаваться так: */
/*      MSC/AL DIAL; */
/*      LINK DIAL+LOW; */
/*-----*/

#include <stdio.h>
#include <dos.h>
#include "user.h"

/* расширенные коды: */
#define KU 72 /* стрелка вниз */

```

```

#define KCR 28 /* возврат каретки */
#define KD 80 /* стрелка вверх */
#define KR 77 /* стрелка вправо */
#define KL 75 /* стрелка влево */
/* программные константы */
#define LEN 12 /* ширина окна меню */
#define NVAR 5 /* число вариантов в меню */
#define RIGHT 2 /* сдвиг подменю относительно основного меню */
#define DOWN 1 /* сдвиг подменю относительно основного меню */
#define TCOLOR 32 /* цвет меню */
#define MCOLOR 64 /* цвет выделенного варианта */
main(){
    s_poscur ( 25, 0); /* спрятать курсор */
    menu( 1,1); /* переход к работе в меню */
    /* меню показывается, начиная со строки 1 и столбца 1 */
}
menu( line, column) int line,column;
{
    int work; /* продолжать работу в текущем меню */
    int variant; /* текущий вариант */
    int scroll; /* включен режим ScrollLock */
    int tlin, tcol; /* положение текущего меню на экране */
    char c; /* расширенный код последнего из введенных символов */
    char buf1[2*LEN*(NVAR+2)];
    char buf2[2*LEN*(NVAR+2)];
    /* буферные массивы для создания эффекта наложения окна */
    tlin = line; tcol = column;
    /* текущее положение меню равно заданному */
    save_rect (tlin, tcol, NVAR+2, LEN, buf1);
    /* в буфере 1 - часть экрана, которая будет закрыта меню */
    show_line (tlin , tcol, " ");
    show_line (tlin+1, tcol, " Меню A ");

```

```

show_line (tlin+2, tcol, " Меню      В ");
show_line (tlin+3, tcol, " Действие А ");
show_line (tlin+4, tcol, " Действие В ");
show_line (tlin+5, tcol, " Выход      ");
show_line (tlin+6, tcol, "              ");

/* на экране нарисовано меню */
draw_rect (tlin, tcol, NVAR+2, LEN, TCOLOR);

/* меню окрашено цветом TCOLOR */
save_rect (tlin, tcol, NVAR+2, LEN, buf2);

/* в буфере 2 - меню, целиком закрашенное в один цвет */
work = 1; variant = 5;

/* заданы текущий вариант и продолжение работы */
while ( work){
    rest_rect ( tlin, tcol, NVAR+2, LEN, buf2);
    /* на экране - одноцветное меню */
    draw_rect ( tlin+variant, tcol+1, 1, LEN-2, MCOLOR);
    /* на экране - меню, в котором */
    /* текущий вариант выделен цветом MCOLOR */
    while(!kbhit()){;}
    /* нажата клавиша */
    c = k_extc(); k_prop();
    scroll = k_reg()&16;
    /* C = расширенный код нажатой клавиши */
    /* sclorr = 1, если включен соответствующий режим */
    if (c == KU && !scroll && variant > 1 ){
        /* вариантом выше */
        variant = variant - 1;
    }else if (c == KD && !scroll && variant < NVAR ){
        /* вариантом ниже */
        variant = variant + 1;
    }else if (c == KU && scroll && tlin > 0 ){
        rest_rect( tlin, tcol, NVAR+2, LEN, buf1);
    }
}

```

```

tlin = tlin - 1;
save_rect (tlin, tcol, NVAR+2, LEN, buf1);
/* в буфере 1 - часть экрана, которая
   будет закрыта сдвинутым меню */
}else if (c == KD && scroll && tlin+NVAR+2 < 25){
    rest_rect( tlin, tcol, NVAR+2, LEN, buf1);
    tlin = tlin + 1;
    save_rect (tlin, tcol, NVAR+2, LEN, buf1);
    /* в буфере 1 - часть экрана, которая
       будет закрыта сдвинутым меню */
}else if (c == KR && scroll && tcol+LEN < 80){
    rest_rect( tlin, tcol, NVAR+2, LEN, buf1);
    tcol = tcol + 1;
    save_rect (tlin, tcol, NVAR+2, LEN, buf1);
    /* в буфере 1 - часть экрана, которая
       будет закрыта сдвинутым меню */
}else if (c == KL && scroll && tcol > 0){
    rest_rect( tlin, tcol, NVAR+2, LEN, buf1);
    tcol = tcol - 1;
    save_rect (tlin, tcol, NVAR+2, LEN, buf1);
    /* в буфере 1 - часть экрана, которая
       будет закрыта сдвинутым меню */
}else if( c == KCR){ /* выбран вариант: */
    if ( variant == 1 ){ /* уход на подменю */
        menu ( line+DOWN, column+RIGHT);

        /* в данном случае вызывается та же процедура.
           В реальной программе, как правило, вызываются
           другие вспомогательные процедуры,
           обеспечивающие работу со своими меню */
    }else if( variant == 5){/* выход */
        work = 0; /* сигнализировать об окончании
                   работы с данным меню */
    }

```

```

        }else{ /* выбран иной, еще не реализованный
                вариант */
                putchar(7); /* звóнок */
        }
    }

    rest_rect( tlin, tcol, NVAR+2, LEN, buf1);

    /* экран приобрел тот же вид, что имел перед вызовом */
    /* этой процедуры */

}

/* изобразить на экране строку */
show_line ( line, column, string )
char string[]; /* что_изобразить */
int line, column; /* с какого места экрана */
{
    int pos;
    pos = 0;
    while (string[pos] != 0){
        VMO [line][column+pos].s = string [pos];
        pos = pos + 1;
    }
}

/* закрасить прямоугольник */
draw_rect( line, column, nline, ncolumn, color)
int line, column, nline, ncolumn, color;
/* строка, столбец, высота, ширина, чем_закрасить */
{
    int l, c;
    s_off();
    for( l = 0; l < nline; l++){

```

```

        for( c = 0; c < ncolumn; c++){
            VMO[line+1][column+c].m = color;
        }
    }
    s_on();
}

/* запомнить прямоугольник */
save_rect( line, column, nline, ncolumn, buf)
int line, column, nline, ncolumn; char buf[];
/* строка, столбец, высота, ширина, где_запомнить */
{
    int l, b, portion;
    s_off();
    b = 0; portion = 2*ncolumn;
    for( l = 0; l < nline; l++){
        memcpy ( &buf[b], &VMO[line+1][column], portion);
        b = b + portion;
    }
    s_on();
}

/* вспомнить прямоугольник */
rest_rect( line, column, nline, ncolumn, buf)
int line, column, nline, ncolumn; char buf[];
/* строка, столбец, высота, ширина, откуда_вспомнить */
{
    int l, b, portion;
    s_off();
    b = 0; portion = 2*ncolumn;
    for( l = 0; l < nline; l++){
        memcpy ( &VMO[line+1][column], &buf[b], portion);
        b = b + portion;
    }
    s_on();
}

```

СПИСОК ЛИТЕРАТУРЫ

1. **Персональные компьютеры.** Информатика для всех. — М.: Наука, 1987. — 149 с.
2. **Трейстер Р.** Персональный компьютер фирмы IBM: Пер. с англ. — М.: Мир, 1986. — 202 с.
3. **Современный компьютер/Пер. с англ. под ред. В. М. Курочкина.** — М.: Мир, 1986. — 210 с.
4. Lord K. W. Using the IBM XT Personal Computer — N. Y.: J. Wiley & Sons. — 1985. — 230 p.
5. **Маклеод Д.** Персональный компьютер, расширяющий возможности контрольно-измерительных и испытательных систем//Электроника. — 1986. — Т. 59, № 6. — С. 52—68.
6. **Саут Х. М., Долсек К. Э.** Обработка сигналов и решение инженерных задач на персональных ЭВМ//ТИИЭР. — 1985. — Т. 73, № 12. — С. 150—163.
7. **Гольденберг Л. М., Матюшкин Б. Д., Поляк М. Н.** Цифровая обработка сигналов. — М.: Радио и связь, 1985. — 312 с.
8. Programs for Digital Signal Processing. — N. Y.: IEEE Press. 1979.
9. Baras E. M. Guide to Using Lotus 1-2-3. — N. Y.: McGraw Hill Books, 1986. — 280 p.
10. **Сиббалд К. Е.** Автоматизированное проектирование и изготовление чертежей на персональных ЭВМ//ТИИЭР. — 1985. — Т. 73, № 12. — С. 128—139.
11. **Ойхман Г. Е.** Графические системы для СМ ЭВМ. — М.: Наука, 1986. — 192 с.
12. Mechanical Engineer Software Guide. — CAE Consultants Press, 1985. — 300 p.
13. **Пайн Р. А., Уокер Р., Гупта Н.** Автоматизированная система разработки на основе пакета программ Matrix/PC. Моделирование и оптимизация на персональных ЭВМ//ТИИЭР. — 1985. — Т. 73, № 12. — С. 139—150.
14. **Уокер Р. А., Шах С., Гупта Н. К.** Автоматизированная разработка систем как метод системного анализа//ТИИЭР. — 1984. — Т. 72, № 12. — С. 77—95.
15. Jones E. Using the dBASE III Plus. — N. Y.: McGraw Hill Books, 1986. — 200 p.
16. Chirlian B. S. Simply dBASE II. — N. Y.: McGraw Hill Books, 1985. — 260 p.
17. Hecht M. File and Data-Base Management Programs for the IBM PC. — N. Y.: J. Wiley & Sons, 1985. — 273 p.
18. **Конопасек М., Джаяраман С.** Языки ограничений и декларативные языки для инженерных приложений. Система TK!SOLVER//ТИИЭР. — 1985. — Т. 73, № 12. — С. 109—128.
19. Konopasek M., Jayaraman S. The TK!SOLVER Book: A Guide to Problem Solving in Science, Engineering, Business and Education. — Berkley: Osborne/McGraw Hill, 1984. — 312 p.
20. Wright V. E. TK!Solver for Engineers. — Reston Publ., 1984. — 277 p.
21. **Логическое проектирование БИС/В. А. Мищенко, А. И. Аспидов, В. В. Витер и др.; Под ред. В. А. Мищенко.** — М.: Радио и связь, 1984. — 312 с.
22. **Реальность и прогнозы искусственного интеллекта/Пер. с англ. под ред. В. Л. Стефанюка.** — М.: Мир 1987. — 246 с.
23. Anderson D., Gessin J. M., Warren F. PC-DOS Tips and Traps. — N. Y. McGraw Hill Books, 1986. — 278 p.
24. **Грис Д.** Наука программирования: Пер. с англ. — М.: Мир, 1984. — 212 с.
25. **Дейкстра Э.** Дисциплина программирования: Пер. с англ. — М.: Мир, 1978. — 150 с.
26. **Вирт Н.** Систематическое программирование. Введение: Пер. с англ. — М.: Мир, 1977. — 260 с.
27. **Дал О., Дейкстра Э., Хоар К.** Структурное программирование: Пер. с англ. — М.: Мир, 1975. — 180 с.

28. Norton P. Inside the IBM PC: Access to Advanced Features and Programming. — N. Y.: J. Wiley & Sons, 1982. — 320 p.
29. Norton P. IBM PC: Programmers Guide. — N. Y.: J. Wiley & Sons. — 1985. — 290 p.
30. Кушниренко Н. Г., Варсанофьев Д. В. Проектирование диалоговых систем: нетрадиционный подход. — М.: Изд-во Моск. гос. ун-та, 1985. — 120 с.
31. Иенсен К., Вирт Н. Паскаль. Руководство для пользователя и описание языка: Пер. с англ. — М.: Финансы и статистика, 1972 г. — 180 с.
32. Conlain J. IBM PC Pascal. — N. Y.: McGraw Hill Books, 1984. — 300 p.
33. Graham N. Programming the IBM PC: Pascal. — Reston, 1983. — 315 p.
34. Керниган Б., Ритчи Д. Язык программирования СИ: Пер. с англ. — М.: Финансы и статистика, 1985. — 270 с.
35. Хэнкок Л., Кригер М. Введение в программирование на языке Си: Пер с англ. — М.: Радио и связь, 1986. — 192 с.
36. Майерс Г. Надежность программного обеспечения: Пер. с англ. — М.: Мир, 1979. — 192 с.
37. Брябрин В. М., Ландау И. Я., Неменман М. Е. О системе кодирования для персональных ЭВМ//Микропроцессорные средства и системы. — 1986. — № 4, С. 61—63.

О Г Л А В Л Е Н И Е

	Стр.
Предисловие	3
Г л а в а 1. Введение в персональную ЭВМ	
1.1. Как ПЭВМ выглядит и из чего состоит	6
1.2. Что такое ДОС и зачем она нужна	7
1.3. Создание текста или оптимальная пишущая машинка	14
1.4. Как заставить ПЭВМ выполнять программы и как научиться с ними работать	24
Г л а в а 2. Инженерно-технические приложения персональной ЭВМ: пакеты программ и работа с ними	
2.1. Автоматизация и управление процессами	46
2.2. Анализ экспериментальных данных	57
2.3. Математические расчеты	75
2.4. Машинная графика	84
2.5. Системы автоматизированного проектирования	95
2.6. Моделирование процессов	121
2.7. Базы данных	134
2.8. Базы знаний и экспертные системы	144
2.9. Интегрированные системы	152
336	

Г л а в а 3. Операционная система, структура и тестирование Стр.
персональной ЭВМ

3.1. Операционная система ДОС, ее структура и возможности	158
3.2. Диалоговые надстройки над ДОС	165
3.3. Работа с пакетными файлами ДОС	176
3.4. Как задать конфигурацию ПЭВМ	181
3.5. Работа с тестирующими программами ПЭВМ	184

Г л а в а 4. Программирование без персональной ЭВМ

4.1. Кем командует программа	189
4.2. Устройство вывода: видеомонитор	191
4.3. Переменные и как ими управлять	194
4.4. Ввод с клавиатуры, или чего ждет от нас программа	197
4.5. Как мы конструируем программы	199
4.6. Процедуры, или способ определения новых команд	208

Г л а в а 5. Программирование для персональной ЭВМ

5.1. Управление операционной системой	213
5.2. Внешние устройства ПЭВМ с разных точек зрения	221
5.3. Клавиатура	231
5.4. Устройство отображения	239
5.5. Другие внешние устройства ПЭВМ	249

Г л а в а 6. Языки программирования персональной ЭВМ

6.1. Язык Паскаль	254
6.2. Язык Си	271
6.3. Язык Бейсик	293
6.4. Несколько замечаний по поводу языка Фортран	315
Приложение. Тексты программ на языке Си	319
Список литературы	335

1 р. 10 к.